SLICOT Working Note 2001-4

Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part I: One-Sided and Coupled Sylvester-Type Equations [1]

Isak Jonsson and Bo Kågström [2]

April 2001, revised August 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/SLWN2001-4.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-01.05, 2001, ISSN-0348-0542, April 2001

Abstract

Triangular matrix equations appear naturally in estimating the condition numbers of matrix equations and different eigenspace computations, including block-diagonalization of matrices and matrix pairs and computation of functions of matrices. To solve a triangular matrix equation is also a major step in the classical Bartels-Stewart method. We present recursive blocked algorithms for solving one-sided triangular matrix equations, including the continuous-time Sylvester and Lyapunov equations, and a generalized coupled Sylvester equation. The main parts of the computations are performed as level 3 general matrix multiply and add (GEMM) operations. Recursion leads to an automatic variable blocking that has the potential of matching the memory hierarchies of today's HPC systems. Different implementation issues are discussed, including when to end the recursion, the design of optimized superscalar kernels for solving leaf-node triangular matrix equations efficiently, and how parallelism is utilized in our implementations. Uniprocessor and SMP parallel performance results of our recursive blocked algorithms and corresponding routines in the state-of-the-art libraries LAPACK and SLICOT are presented. The performance improvements of our recursive algorithms are remarkable, including 10-folded speedups compared to standard algorithms.

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

Contents

1 Introduction

We introduce and discuss new recursive blocked algorithms for solving various types of triangular matrix equations. Our goal is to design efficient algorithms for state-of-the-art HPC systems with deep memory hierarchies. A recursive algorithm leads to automatic blocking which is variable and "squarish" [13]. This hierarchical blocking allows for good data locality, which makes it possible to approach peak performance on state-of-the-art processors with several levels of cache memories. Recently, several successful results have been published. Using the current standard data layouts and applying recursive blocking have led to faster algorithms for the Cholesky, LU and QR factorizations [13, 8]. Moreover, in [14], we demonstrated that a further performance gain can be obtained when recursive dense linear algebra algorithms are expressed using a recursive data layout and highly optimized superscalar kernels.

In this contribution (Part I), which extends on our work in [22], we consider one-sided Sylvester-type equations, including the continuous-time standard Sylvester ($AX - XB = C$) and Lyapunov ($AX + XA^T = C$) equations, and a generalized coupled Sylvester equation $(AX - YB, DX - YE) = (C, F)$. We use the notation *one-sided*, since the matrix equations include terms where the solution is only involved in matrix products of two matrices (e.g., $\text{op}(A)X$ or $X\text{op}(A)$, where $\text{op}(A)$ can be A or A^T). These equations all appear in various control theory applications and in spectral analysis.

The classical method of solution is the Bartels-Stewart method, which includes three major steps [3]. First, the matrix (or matrix pair) is transformed to a Schur (or generalized Schur) form. This leads to a reduced triangular matrix equation, which is solved in the second step. For example, the coefficient matrices A and B in the Sylvester equation $AX - XB = C$ are in upper triangular or upper quasi-triangular form [31, 45]. For the generalized counterpart, the matrix pairs (A, D) and (B, E) in $(AX - YB, DX - YE) = (C, F)$ are reduced to generalized Schur form, with A and B upper quasi-triangular, and D, E upper triangular [35, 32]. Finally, the solution of the reduced matrix equation is transformed back to the original coordinate system. In this paper, we focus on the solution of the reduced triangular matrix equations. Reliable and efficient algorithms for the reduction step can be found in LAPACK [1] for the standard case, and in [5] for the generalized case, where a blocked variant of the QZ method is presented.

Triangular matrix equations also appear naturally in estimating the condition numbers of matrix equations and different eigenspace computations, including block-diagonalization of matrices and matrix pairs and computation of functions of matrices. Related applications include the direct reordering of eigenvalues in the real (generalized) Schur form [1, 33] and the computation of additive decompositions of a (generalized) transfer function [34].

Before we go into any further details, we outline the contents of the rest of the paper. In Section 2, we illustrate how the solutions of triangular matrix equations relate to different spectral condition estimation problems and to the estimation of the conditioning of the Sylvester-type equations themselves. Section 3 introduces our recursive blocked algorithms for one-sided and coupled triangular matrix equations, including the standard Sylvester (Section 3.1) and Lyapunov (Section 3.2) equations, and the generalized coupled Sylvester equation (Section 3.3). In Section 4, we introduce a recursive blocked algorithm for computing functions of matrices, which, indeed, is an application of solving triangular Sylvester equations.

In Section 5, we discuss different implementation issues, including when to end the recursion, and the design of optimized superscalar kernels for solving small-sized (leaf-node) triangular matrix equations efficiently. We also discuss how parallelism is utilized in our implementations. Sample performance results of our recursive blocked algorithms are presented and discussed in Section 6. Finally, we give some conclusions and outline future work.

2 Condition Estimation and Triangular Solvers

Besides solving a matrix equation it is equally important to have reliable error bounds of the computed quantities. In this section, we review how triangular matrix equations enter in condition estimation of common eigenspace computations, as well as, in the condition estimation of the matrix equations themselves. Since condition estimation typically involves solving several triangular matrix equations, one could say that this is the main source for different triangular matrix equation problems. It is important that these matrix equations can be solved with efficient algorithms on today's memory tiered systems.

2.1 Condition Estimation of Some Eigenspace Problems

We assume that S is a block partitioned matrix in *real Schur form*:

$$S = \begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}.$$

This means that both A and B are *quasi-upper triangular* (or upper quasi-triangular), i.e., block upper triangular with 1×1 and 2×2 diagonal blocks, which correspond to real and complex conjugate pairs of eigenvalues, respectively. Typically, the partitioning is done with some application in mind. For example, A may include all eigenvalues in the left complex plane and we want the invariant subspaces associated with the spectra of A and B, respectively.

Now, S can be block diagonalized by a similarity transformation:

$$\begin{bmatrix} I & -X \\ 0 & I \end{bmatrix} S \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix},$$

where X satisfies the triangular Sylvester equation $AX - XB = C$.

Knowing X, we also know the invariant subspaces and the *spectral projector* associated with the block A :

$$P = \begin{bmatrix} I & X \\ 0 & 0 \end{bmatrix}.$$

This is an important quantity in error bounds for invariant subspaces and clusters of eigenvalues. A large value of $\|P\|_2 = (1 + \|R\|_2^2)^{1/2}$ signals ill-conditioning. To avoid possible overflow, we use the computed estimate $s = 1/\|P\|_F$ [2, 31, 1].

Next, we consider a regular matrix pair (S, T) in *real generalized Schur form*:

$$(S,T) = \left(\begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}, \begin{bmatrix} D & -F \\ 0 & E \end{bmatrix} \right),$$

where A, B, as before, are upper quasi-triangular and D, E are upper triangular. Any 2×2 diagonal block in the generalized Schur form corresponds to a pair of complex conjugate eigenvalues. The 1×1 diagonal blocks correspond to real eigenvalues. For example, if $e_{ii} \neq 0$, then a_{ii}/e_{ii} is a finite eigenvalue of (A, D). Otherwise $(e_{ii} = 0)$, the matrix pair has an infinite eigenvalue.

Now, (S, T) is block diagonalized by an equivalence transformation:

$$\begin{bmatrix} I & -Y \\ 0 & I \end{bmatrix} (S,T) \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} = \left(\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}, \begin{bmatrix} D & 0 \\ 0 & E \end{bmatrix} \right),$$

where (X, Y) satisfies the triangular coupled Sylvester equation $(AX - YB, DX - YE) = (C, F)$. Knowing (X, Y), we also know pairs of deflating subspaces associated with the matrix pairs (A, D) and (B, E). Similarly to the matrix case, large values on the *left* and *right projector norms* $l = (1 + \|Y\|_F^2)^{1/2}$ and $r = (1 + \|X\|_F^2)^{1/2}$ signal ill-conditioning [33, 1], i.e., the eigenspaces may be sensitive to small perturbations in the data.

2.2 Condition Estimation of Matrix Equations

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

We introduce the following Z-matrices:

$$\begin{aligned} Z_{\mathrm{SYCT}} &= Z_{AX-XB} = I_n \otimes A - B^T \otimes I_m, \\ Z_{\mathrm{LYCT}} &= Z_{AX+XA^T} = I_n \otimes A + A \otimes I_n, \\ Z_{\mathrm{GCSY}} &= Z_{(AX-YB,DX-YE)} = \begin{bmatrix} I_n \otimes A & -B^T \otimes I_m \\ I_n \otimes D & -E^T \otimes I_m \end{bmatrix}, \end{aligned}$$

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

An important quantity that both appear in the perturbation theory for Sylvester-type equations and for the eigenspace problems considered above is the *separation between two matrices* [47], defined as

$$\mathrm{Sep}[A, B] = \inf_{\|X\|_F=1} \|AX - XB\|_F = \sigma_{\min}(Z_{\mathrm{SYCT}}),$$

where $\sigma_{\min}(Z_{\mathrm{SYCT}}) \geq 0$ is the smallest singular value of Z_{SYCT}. We review some of its characteristics: $\mathrm{Sep}[A, B] = 0$ if and only if A and B have a common eigenvalue; $\mathrm{Sep}[A, B]$ is small if there is small perturbation of A or B that makes them have a common eigenvalue. The Sep-function may be much smaller than the minimum distance between the eigenvalues of A and B.

Assuming M and N are the dimensions of A and B, computing $\sigma_{\min}(Z_{\mathrm{SYCT}})$ is an $O(M^3N^3)$ operation, which is impractical already for moderate values on M and N. In [31], it is shown how reliable Sep^{-1}-estimates can be computed to the cost $O(MN^2 + M^2N)$ by solving triangular matrix equations:

$$\frac{\|x\|_2}{\|c\|_2} = \frac{\|X\|_F}{\|C\|_F} \leq \|Z^{-1}_{\mathrm{SYCT}}\|_2 = \frac{1}{\sigma_{\min}(Z_{\mathrm{SYCT}})} = \mathrm{Sep}^{-1}.$$

The right hand side C is chosen such that the lower bound gets as large as possible. This leads to a Frobenius-norm-based estimate. For computation of 1-norm-based estimates see [17, 19, 31].

The Sep-functions associated with the Sylvester-type matrix equations are:

$$\begin{aligned} \mathrm{Sep}[\mathrm{SYCT}] &= \inf\nolimits_{\|X\|_F=1} \|AX - XB\|_F &&= \sigma_{\min}(Z_{\mathrm{SYCT}}), \\ \mathrm{Sep}[\mathrm{LYCT}] &= \inf\nolimits_{\|X\|_F=1} \|AX - X(-A^T)\|_F &&= \sigma_{\min}(Z_{\mathrm{LYCT}}), \\ \mathrm{Sep}[\mathrm{GCSY}] &= \inf\nolimits_{\|(X,Y)\|_F=1} \|(AX - YB, DX - YE)\|_F &&= \sigma_{\min}(Z_{\mathrm{GCSY}}). \end{aligned}$$

The same techniques as presented above can also be used for estimating Sep[LYCT]. Realiable estimates of Sep[GCSY] (the separation between two matrix pairs [47]) are presented and discussed in [35, 32, 33]. The underlying perturbation theory for these Sylvester-type equations is presented in [20, 27]. See also the nice review in Chapter 15 of [21].

3 Recursive Blocked Algorithms for One-Sided and Coupled Matrix Equations

As mentioned in the introduction, the standard methods for solving one-sided matrix equations are all based on the Bartels-Stewart method [3]. The fundamental algorithms for solving the continuous-time

Sylvester and Lyapunov equations are presented in [3, 11, 18]. Generalizations of the Bartels-Stewart and the Hessenberg-Schur [11] methods for solving the generalized coupled Sylvester equation can be found in [35, 32].

In this section, we present our recursive blocked methods for solving triangular one-sided and coupled matrix equations, which also are amenable for parallelization, especially on shared memory systems. Parallel methods for solving triangular one-sided and coupled matrix equations have also been studied, e.g., see [31, 45, 44].

For each matrix equation we define recursive splittings which in turn lead to a few smaller problems to be solved. These recursive splittings are applied to all "half-sized" triangular matrix equations and so on. 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 new high-performance kernels based on standard algorithms (see Section 5).

We 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 general matrix multiply and add (GEMM) operations $C \leftarrow \beta C + \alpha AB$, where α and β are real scalars. This is due to the "one-sidedness" of the matrix equations. The function $[C] = \mathbf{gemm}(A, B, C)$ implements the GEMM-operation $C = C + AB$. Other functions, including level 3 BLAS [6, 7, 29, 30] operations are introduced the first time they are used in the algorithm descriptions.

3.1 Recursive triangular continuous-time Sylvester solvers

Consider the real *continuous-time Sylvester* (SYCT) matrix equation

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

where A of size $M \times M$ and B of size $N \times N$ 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 $M \times N$. Typically, the solution overwrites the right hand side ($C \leftarrow X$). The SYCT equation (1) has a *unique solution* if and only if A and B have no eigenvalue in common. This follows immediately from the definition of Sep[SYCT] in Section 2.2.

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

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

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

or equivalently

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

The original problem is split in two triangular Sylvester equations. First, we solve for X_2 and after the GEMM update $C_1 = C_1 - A_{12}X_2$, we can solve for X_1.

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

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

or equivalently

$$\begin{aligned} AX_1 - X_1B_{11} &= C_1, \\ AX_2 - X_2B_{22} &= C_2 + X_1B_{12}. \end{aligned}$$

Now, we first solve for X_1 and then after updating C_2 with respect to X_1, X_2 can be solved for.

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

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

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

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

We start by solving for X_{21} in the third equation. After updating C_{11} and C_{22} with respect to X_{21}, we can solve for X_{11} and X_{22}. Both updates and the triangular Sylvester solves are independent operations and can be executed concurrently. Finally, we update C_{12} with respect to X_{11} and X_{22}, and solve for X_{12}.

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 SYCT, where A and B, each of them can be in either upper or lower Schur form.

A Matlab-style function $[X] = \mathbf{rtrsyct}(A, B, C, uplo, blks)$ implementing our recursive blocked solver is presented in Algorithm 1. The input *uplo* signals the triangular structure of A and B. The function $[X] = \mathbf{trsyct}(A, B, C, uplo)$ implements an algorithm for solving triangular Sylvester block kernel problems (see Section 5).

3.2 Recursive triangular continuous-time Lyapunov solvers

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

$$AX + XA^T = C, \tag{2}$$

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 ($C \leftarrow X$). The LYCT equation (2) has a *unique solution* if and only if $\lambda_i(A) + \lambda_j(A) \neq 0$ for all i and j. This follows immediately from the definition of Sep[LYCT] in Section 2.2. If $C = C^T$ is (semi)definite and $\text{Re}\lambda_i(A) < 0$ for all i, then a unique (semi)definite solution X exists [18].

Since all matrices are of the same size, there is only one way of doing the *recursive splitting*. We split A and C by rows and columns:

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

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

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

The first and the third are triangular Lyapunov equations, while the second is a triangular Sylvester matrix equation. We start by solving for X_{22} in the third equation. After updating C_{12} with respect

Algorithm 1: rtrsyct

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 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 $AX - XB = C$. X is allowed to overwrite C.

```
function $[X] = \mathbf{rtrsyct}(A, B, C, uplo, blks)$
if $1 \leq M, N \leq blks$ then
    $X = \mathbf{trsyct}(A, B, C, uplo)$;
else  switch $uplo$
case 1
    if $1 \leq N \leq M/2$   % Case 1: Split $A$ (by rows and colums), $C$ (by rows only)
        $X_2 = \mathbf{rtrsyct}(A_{22}, B, C_2, 1, blks)$;
        $C_1 = \mathbf{gemm}(-A_{12}, X_2, C_1)$;
        $X_1 = \mathbf{rtrsyct}(A_{11}, B, C_1, 1, blks)$;
        $X = [X_1; X_2]$;
    elseif $1 \leq M \leq N/2$   % Case 2: Split $B$ (by rows and colums), $C$ (by columns only)
        $X_1 = \mathbf{rtrsyct}(A, B_{11}, C_1, 1, blks)$;
        $C_2 = \mathbf{gemm}(X_1, B_{12}, C_2)$;
        $X_2 = \mathbf{rtrsyct}(A, B_{22}, C_2, 1, blks)$;
        $X = [X_1, X_2]$;
    else   % $M, N \geq blks$, Case 3: Split $A$, $B$ and $C$ (all by rows and colums)
        $X_{21} = \mathbf{rtrsyct}(A_{22}, B_{11}, C_{21}, 1, blks)$;
        $C_{22} = \mathbf{gemm}(X_{21}, B_{12}, C_{22})$; $C_{11} = \mathbf{gemm}(-A_{12}, X_{21}, C_{11})$;
        $X_{22} = \mathbf{rtrsyct}(A_{22}, B_{22}, C_{22}, 1, blks)$; $X_{11} = \mathbf{rtrsyct}(A_{11}, B_{11}, C_{11}, 1, blks)$;
        $C_{12} = \mathbf{gemm}(-A_{12}, X_{22}, C_{12})$;
        $C_{12} = \mathbf{gemm}(X_{11}, B_{12}, C_{12})$;
        $X_{12} = \mathbf{rtrsyct}(A_{11}, B_{22}, C_{12}, 1, blks)$;
        $X = [X_{11}, X_{12};\ X_{21}, X_{22}]$;
    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 continuous-time Sylvester equation.

to X_{22}, we can solve for X_{12}. Finally, we update the right hand side of the first matrix equation with respect to X_{12}, which is a symmetric rank-$2k$ (SYR2K) operation $C_{11} = C_{11} - A_{12}X_{12}^T - X_{12}A_{12}^T$, and solve for X_{11}.

In Algorithm 2, we present a Matlab-style function $[X] = \mathbf{rtrlyct}(A, C, blks)$ implementing our recursive blocked solver, which deals with the cases with a symmetric or nonsymmetric C. The function $[C] = \mathbf{syr2k}(A, B, C)$ implements the SYR2K-operation $C = C + AB^T + BA^T$. The function $[X] = \mathbf{trlyct}(A, C)$ implements an algorithm for solving triangular Lyapunov block kernel problems (see Section 5). For solving the triangular Sylvester equations that appear, we make use of the recursive algorithm $[X] = \mathbf{rtrsyct}(A, B, C, uplo, blks)$, described in Section 3.1.

Algorithm 2: rtrlyct

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 triangular matrix equations.

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

```
function $[X] = \mathbf{rtrlyct}(A, C, blks)$
if $1 \leq N \leq blks$ then
    $X = \mathbf{trlyct}(A, C)$;
elseif $C$ is symmetric
    % Split $A$ and $C$ (by rows and colums)
    $X_{22} = \mathbf{rtrlyct}(A_{22}, C_{22}, blks)$;
    $C_{12} = \mathbf{gemm}(-A_{12}, X_{22}, C_{12})$;
    $X_{12} = \mathbf{rtrsyct}(A_{11}, -A_{22}^T, C_{12}, 2, blks)$;  $X_{21} = X_{12}^T$;
    $C_{11} = \mathbf{syr2k}(-A_{12}, X_{12}, C_{11})$;
    $X_{11} = \mathbf{rtrlyct}(A_{11}, C_{11}, blks)$;
    $X = [X_{11}, X_{12};\ X_{21}, X_{22}]$;
else
    $X = \mathbf{rtrsyct}(A, -A^T, C, 2, blks)$;
end
```

Algorithm 2: Recursive blocked algorithm for solving the triangular continuous-time Lyapunov equation.

3.3 Recursive triangular coupled Sylvester solvers

Consider the real *generalized coupled Sylvester* (GCSY) matrix equation of the form

$$\begin{array}{rcl} AX - YB & = & C, \quad C \leftarrow X \ (M \times N), \\ DX - YE & = & F, \quad F \leftarrow Y \ (M \times N), \end{array} \tag{3}$$

where the matrix pairs (A, D) and (B, E) are in generalized Schur form with A, B (quasi-)upper triangular and D, E upper triangular. A, D are $M \times M$ and B, E are $N \times N$; the right hand sides C, D and the solution matrices X, Y are of size $M \times N$. The GCSY equation (3) has a *unique solution* if and only if (A, D) and (B, E) are regular matrix pairs and have no eigenvalue in common. This follows immediately from the definition of Sep[GCSY] in Section 2.2.

As for the standard case, we consider three alternatives for doing a *recursive splitting*. Below, we illustrate these cases.

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

$$\begin{array}{rcl} \begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} B & = & \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}, \\ \begin{bmatrix} D_{11} & D_{12} \\ & D_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} E & = & \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}. \end{array}$$

The splitting results in the following two generalized Sylvester equations:

$$\begin{array}{rcl} A_{11}X_1 - Y_1B & = & C_1 - A_{12}X_2, \\ D_{11}X_1 - Y_1E & = & F_1 - D_{12}X_2, \\ A_{22}X_2 - Y_2B & = & C_2, \\ D_{22}X_2 - Y_2E & = & F_2. \end{array}$$

First, we solve for (X_2, Y_2) in the second pair of matrix equations. After updating (C_1, F_1) with respect to X_2 (two GEMM operations that can execute in parallel), we solve for (X_1, Y_1).

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

$$\begin{array}{rcl} A \left[\begin{array}{cc} X_1 & X_2 \end{array} \right] - \left[\begin{array}{cc} Y_1 & Y_2 \end{array} \right] \left[\begin{array}{cc} B_{11} & B_{12} \\ & B_{22} \end{array} \right] & = & \left[\begin{array}{cc} C_1 & C_2 \end{array} \right], \\ D \left[\begin{array}{cc} X_1 & X_2 \end{array} \right] - \left[\begin{array}{cc} Y_1 & Y_2 \end{array} \right] \left[\begin{array}{cc} E_{11} & E_{12} \\ & E_{22} \end{array} \right] & = & \left[\begin{array}{cc} F_1 & F_2 \end{array} \right], \end{array}$$

or equivalently

$$\begin{array}{rcl} AX_1 - Y_1 B_{11} & = & C_1, \\ DX_1 - Y_1 E_{11} & = & F_1, \\ AX_2 - Y_2 B_{22} & = & C_2 + Y_1 B_{12}, \\ DX_2 - Y_2 E_{22} & = & F_2 + Y_1 E_{12}. \end{array}$$

Now we first solve for (X_1, Y_1), and after updating (C_2, F_2) with respect to Y_1 (also two independent GEMM operations), we solve for (X_2, Y_2).

Case 3 $(N/2 \le M \le 2N)$. We split all matrices by rows and columns:

$$\begin{array}{rcl} \left[\begin{array}{cc} A_{11} & A_{12} \\ & A_{22} \end{array} \right] \left[\begin{array}{cc} X_{11} & X_{12} \\ X_{21} & X_{22} \end{array} \right] - \left[\begin{array}{cc} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{array} \right] \left[\begin{array}{cc} B_{11} & B_{12} \\ & B_{22} \end{array} \right] & = & \left[\begin{array}{cc} C_{11} & C_{12} \\ C_{21} & C_{22} \end{array} \right], \\ \left[\begin{array}{cc} D_{11} & D_{12} \\ & D_{22} \end{array} \right] \left[\begin{array}{cc} X_{11} & X_{12} \\ X_{21} & X_{22} \end{array} \right] - \left[\begin{array}{cc} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{array} \right] \left[\begin{array}{cc} E_{11} & E_{12} \\ & E_{22} \end{array} \right] & = & \left[\begin{array}{cc} F_{11} & F_{12} \\ F_{21} & F_{22} \end{array} \right]. \end{array}$$

This leads to four pairs of coupled Sylvester equations to be solved:

$$\begin{array}{rcl} A_{11}X_{11} - Y_{11}B_{11} & = & C_{11} - A_{12}X_{21}, \\ D_{11}X_{11} - Y_{11}E_{11} & = & F_{11} - D_{12}X_{21}, \\ A_{11}X_{12} - Y_{12}B_{22} & = & C_{12} - A_{12}X_{22} + Y_{11}B_{12}, \\ D_{11}X_{12} - Y_{12}D_{22} & = & F_{12} - D_{12}X_{22} + Y_{11}E_{12}, \\ A_{22}X_{21} - Y_{21}B_{11} & = & C_{21}, \\ D_{22}X_{21} - Y_{21}E_{11} & = & F_{21}, \\ A_{22}X_{22} - Y_{22}B_{22} & = & C_{22} + Y_{21}B_{12}, \\ D_{22}X_{22} - Y_{22}E_{22} & = & F_{22} + Y_{21}E_{12}. \end{array}$$

We start by solving for (X_{21}, Y_{21}) in the third matrix equation pair. After updating (C_{11}, F_{11}) and (C_{22}, F_{22}) with respect to X_{21} and Y_{21}, respectively, we can solve for (X_{11}, Y_{11}) and (X_{22}, Y_{22}). Finally, we update (C_{12}, F_{12}) with respect to Y_{11} and X_{22}, and solve for (X_{12}, Y_{12}). Several of these operations can be performed concurrently, including GEMM-updates of right hand sides and two of the generalized triangular Sylvester solves.

A Matlab-style function, $[X] = \mathbf{rtrgcsy}(A, B, C, D, E, F, blks)$, which implements our recursive blocked generalized Sylvester algorithm is presented in Algorithm 3. As in the standard case, the recursion is only applied down to a certain block size, when the function $[X] = \mathbf{trgcsy}(A, B, C, D, E, F)$ is applied for solving triangular generalized coupled Sylvester block kernel problems.

3.4 Number of operations and execution order

All recursive algorithms presented in Section 3, perform the same amount of *floating point operations* (flops) as the elementwise explicit algorithms or their blocked counterparts (e.g., see [3, 35, 31, 32]), which

Algorithm 3: rtrgcsy

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

Output: (X, Y) of size $M \times N$, the solution of $(AX - YB, DX - YE) = (C, F)$. X, Y are allowed to overwrite C, F.

```
function $[X, Y] = \mathbf{rtrgcsy}(A, B, C, D, E, F, blks)$
if $1 \le M, N \le blks$ then
    $[X, Y] = \mathbf{trgcsy}(A, B, C, D, E, F)$;
elseif $1 \le N \le M/2$   % Case 1: Split $(A, D)$ (by rows and colums), $(C, F)$ (by rows only)
    $[X_2, Y_2] = \mathbf{rtrgcsy}(A_{22}, B, C_2, D_{22}, E, F_2, blks)$;
    $C_{11} = \mathbf{gemm}(-A_{12}, X_2, C_1)$; $F_{11} = \mathbf{gemm}(-D_{12}, X_2, F_1)$;
    $[X_1, Y_1] = \mathbf{rtrgcsy}(A_{11}, B, C_1, D_{11}, E, F_1, blks)$;
    $X = [X_1; X_2]$; $Y = [Y_1; Y_2]$;
elseif $1 \le M \le N/2$   % Case 2: Split $(B, E)$ (by rows and colums), $(C, F)$ (by columns only)
    $[X_1, Y_1] = \mathbf{rtrgcsy}(A, B_{11}, C_1, D, E_{11}, F_1, blks)$;
    $C_2 = \mathbf{gemm}(Y_1, B_{12}, C_2)$; $F_2 = \mathbf{gemm}(Y_1, E_{12}, F_2)$;
    $[X_2, Y_2] = \mathbf{rtrgcsy}(A, B_{22}, C_2, D, E_{22}, F_2, blks)$;
    $X = [X_1, X_2]$; $Y = [Y_1, Y_2]$;
else   % $M, N \ge blks$, Case 3: Split $(A, D)$, $(B, E)$ and $(C, F)$ (all by rows and colums)
    $[X_{21}, Y_{21}] = \mathbf{rtrgcsy}(A_{22}, B_{11}, C_{21}, D_{22}, E_{11}, F_{21}, blks)$;
    $C_{22} = \mathbf{gemm}(Y_{21}, B_{12}, C_{22})$; $C_{11} = \mathbf{gemm}(-A_{12}, X_{21}, C_{11})$;
    $F_{22} = \mathbf{gemm}(Y_{21}, E_{12}, F_{22})$; $F_{11} = \mathbf{gemm}(-D_{12}, X_{21}, F_{11})$;
    $[X_{22}, Y_{22}] = \mathbf{rtrgcsy}(A_{22}, B_{22}, C_{22}, D_{22}, E_{22}, F_{22}, blks)$;
    $[X_{11}, Y_{11}] = \mathbf{rtrgcsy}(A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, blks)$;
    $C_{12} = \mathbf{gemm}(-A_{12}, X_{22}, C_{12})$; $F_{12} = \mathbf{gemm}(-D_{12}, X_{22}, F_{12})$;
    $C_{12} = \mathbf{gemm}(Y_{11}, B_{12}, C_{12})$; $F_{12} = \mathbf{gemm}(Y_{11}, E_{12}, F_{12})$;
    $[X_{12}, Y_{12}] = \mathbf{rtrgcsy}(A_{11}, B_{22}, C_{12}, D_{11}, E_{22}, F_{12}, blks)$;
    $X = [X_{11}, X_{12}; X_{21}, X_{22}]$; $Y = [Y_{11}, Y_{12}; Y_{21}, Y_{22}]$;
end
```

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

are all based on backward or forward substitutions with one or several right hand sides. In Table 1, we summarize the overall flopcounts for these methods. 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, does only show up in the lower order terms.

Matrix equation	Overall cost in flops
SYCT (1)	$M^2N + MN^2$
LYCT (2)	N^3
GCSY (3)	$2M^2N + 2MN^2$

Table 1: Complexity of standard algorithms measured in flops.

The flopcounts for the recursive blocked algorithms can be expressed in terms of the following recurrence formulas:

$$\begin{aligned} F_{\mathrm{SYCT}}(M, N) &= 4F_{\mathrm{SYCT}}(M/2, N/2) + 2F_{\mathrm{GEMM}}(M/2, M/2, N/2) \\ &\quad +2F_{\mathrm{GEMM}}(M/2, N/2, N/2), \end{aligned} \tag{4}$$

$$F_{\mathrm{LYCT}}(N) = 2F_{\mathrm{LYCT}}(N/2) + F_{\mathrm{SYCT}}(N/2, N/2) + 2F_{\mathrm{GEMM}}(N/2, N/2, N/2), \qquad (5)$$

$$\begin{aligned} F_{\mathrm{GCSY}}(M, N) &= 4F_{\mathrm{GCSY}}(M/2, N/2) + 4F_{\mathrm{GEMM}}(M/2, M/2, N/2) \\ &\quad +4F_{\mathrm{GEMM}}(M/2, N/2, N/2). \end{aligned} \qquad (6)$$

These expressions correspond to the most general case when the recursive splitting is by rows and by columns for all input matrices. Ignoring the lower order terms and assuming that, $F_{\mathrm{GEMM}}(P, Q, R)$, the complexity of the GEMM operation with matrices of sizes $P \times Q$ and $Q \times R$ is $2PQR$ flops, it is straightforward to derive the expressions in Table 1 by induction from the equations (4), (5), and (6), respectively.

The main difference between the recursive blocked and the standard explicitly blocked algorithms is 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. As is shown in Section 6, this can have a great impact on the performance of the algorithms. As expected, the algorithms with the smallest amount of redundant memory transfers show the best performance.

4 Computing Functions of Triangular Matrices Using Recursive Blocking

An important application to solving triangular Sylvester equations is the problem of computing $\mathbf{f}(A)$, where $\mathbf{f}$ is an analytic function and A of size $N \times N$ is a real matrix in Schur form, e.g., A is quasi-upper triangular. The most well-known matrix function is the matrix exponential, which has several applications in control theory. Different methods have been suggested over the years for computing matrix functions. We refer to the papers [24, 25, 48, 40], which also include perturbation theory and error bounds. Several of these results are also reviewed in [12], the standard textbook on advanced matrix computations.

In the following, we let F denote the matrix function $\mathbf{f}(A)$ to be computed. Since A is upper triangular, so will F be. Moreover, since $\mathbf{f}$ is analytic F can be expressed in terms of a series expansion, from which it is obvious that F and A commute, i.e.,

$$AF - FA = 0. \qquad (7)$$

We use this fact to derive a blocked recursive algorithm for computing F. Since A is square, there is only one way of doing the *recursive splitting*, namely, we split A and F by rows and columns:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} \\ & F_{22} \end{bmatrix} - \begin{bmatrix} F_{11} & F_{12} \\ & F_{22} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

This recursive splitting results in the following three triangular matrix equations:

$$\begin{aligned} A_{11}F_{11} - F_{11}A_{11} &= 0 \\ A_{11}F_{12} - F_{12}A_{22} &= F_{11}A_{12} - A_{12}F_{22}, \\ A_{22}F_{22} - F_{22}A_{22} &= 0 \end{aligned}$$

The first and the third reveal the commuting properties of the diagonal blocks. Knowing F_{11} and F_{22}, we get F_{12} from the second equation, which is a triangular continuous-time Sylvester equation (1).

The splitting described above can now be applied recursively to all "half-sized" triangular matrix equations. As before, we end the recursion when a problem size N is smaller than a certain block size, $blks$. For computing the matrix function of the small-sized kernel problems, there are several alternatives.

Algorithm 4: rtrmfcn

Input: A ($N \times N$) in upper quasi-triangular (Schur) form. fcn, analytic function. $blks$, block size that specifies when to switch to a standard algorithm for computing $fcn(A)$.

Output: F ($N \times N$), the computed $fcn(A)$.

```
function $[F] = \textbf{rtrmfcn}(A, fcn, blks)$
if $1 \le N \le blks$ then
    $X = \textbf{trmfcn}(A, fcn)$;
else    % Split $A$ (by rows and colums)
    $F_{11} = \textbf{rtrmfcn}(A_{11}, fcn, blks)$; $F_{22} = \textbf{rtrmfcn}(A_{22}, fcn, blks)$;
    $F_{12} = \textbf{trmm}(F_{11}, A_{12}) + \textbf{trmm}(-A_{12}, F_{22})$;  $\%F_{11}A_{12} - A_{12}F_{22}$
    $F_{12} = \textbf{rtrsyct}(A_{11}, -A_{22}, F_{12}, blks)$;
    $F = [F_{11}, F_{12};\ 0, F_{22}]$;
end
```

Algorithm 4: Recursive blocked algorithm for computing triangular matrix functions.

If all eigenvalues are real, one alternative is to choose $blks = 1$, which means that the recursion continues until element level and $F_{ii} = f(\lambda_j)$ for an eigenvalue λ_j.

However, in this context, the choice of block size is also important with respect to the accuracy and reliability of the results. If the eigenvalues along the (block) diagonal of A are ordered with respect to a clustering procedure, then the blocking with respect to the clustering must be taken into account in the recursive splitting. Splittings that are not permitted can be monitored by computing an estimate of Sep$[A_{11}, A_{22}]$, each time a Sylvester equation is solved for computing an F_{12} block. A small value of Sep indicates that a small perturbation of A_{11} and/or A_{22} may cause at least one eigenvalue from the perturbed A_{11} to coalesce with an eigenvalue of A_{22}. If this is the case, the splitting of A is inadmissible and another splitting should be chosen. To facilitate the choice of splitting, the blocking with respect to clustering should be provided as input. There is a lot more to say, but this is out of the scope of this paper.

In Algorithm 4, a Matlab-style function $[F] = \textbf{rtrmfcn}(A, fcn, blks)$ for our blocked recursive algorithm to compute $fcn(A)$ is presented. The function $[F] = \textbf{trmfcn}(A, fcn)$ implements a standard algorithm for computing $F = fcn(A)$. For solving the F_{12} blocks, we apply the recursive Sylvester algorithm $[X] = \textbf{rtrsyct}(A, B, C, blks)$, described in Section 3.1. For brevity, we have omitted the monitoring of inadmissible recursive splittings in this description.

5 Optimized Superscalar Kernels and Other Implementation Issues

In principle, we have three levels of triangular matrix equation solvers. At the user-level we have 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 $M, N \le 4$. There are several issues to consider in developing portable and high-performance implementations for these solvers.

In this section, we discuss several of them including the impact of kernel solvers to the overall performance of the matrix equation solver, the design of superscalar kernels for solving small-sized matrix equations, and when to end the recursion and instead call a block solver for solving the remaining sub-systems. We also discuss different aspects in the choice of BLAS implementations and in implementing shared memory parallelism.

5.1 Impact of kernel solvers

For illustration and without loss of generality, we assume that all matrix sizes are the same ($M = N$). Then the innermost kernels of the Sylvester-type solvers, which exist in the **tr*** routines of the algorithm descriptions in Section 3, execute $O(N^2)$ flops out of the total $O(N^3)$ flops. Though one order less operations, the performance of the kernels are very important for the overall computation rate for these solvers. Most of the operations outside the kernels are GEMM operations.

In Figure 1, the modeled impact of the performance of Sylvester kernels on the overall performance of the recursive blocked algorithm **rtrsyct** is illustrated. The x-axis shows the problem size N, and the y-axis shows the overall performance of **rtrsyct** measured in Mflops/s for kernel routines with different performance characteristics. We use two fixed performance rates for the DGEMM routine, 200 and 500 Mflops/s, respectively. With $M = N$, the total number of flops is $2N^3$, and the number of flops performed by the 2×2 sub-system kernel is $4N^2$. We model the *overall computation rate* as

$$S = \frac{\text{Total number of flops}}{\text{GEMM time} + \text{Kernel time}} = \frac{2N^3}{\frac{2N^3-4N^2}{G} + \frac{4N^2}{K}} = \frac{N \cdot G}{N - 2 + 2\frac{G}{K}}, \tag{8}$$

where G denotes the performance of the DGEMM routine, and K denotes the performance of the kernel routine.

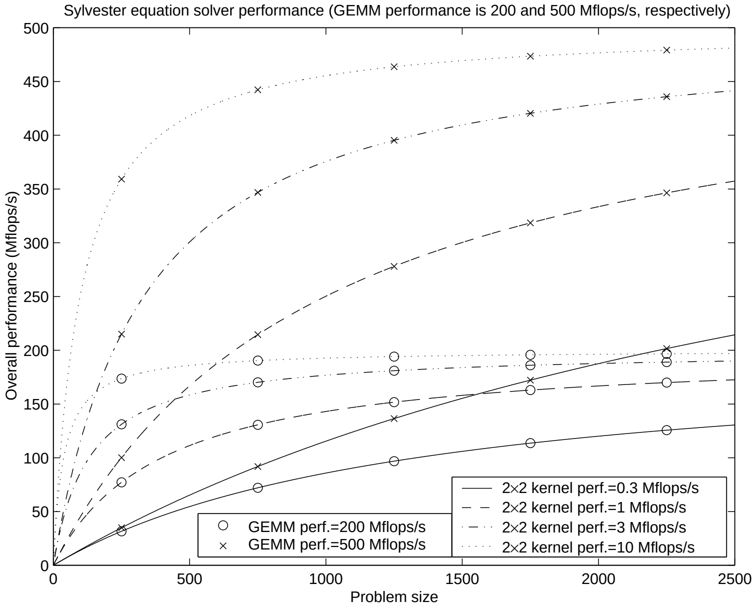


Figure 1: Modeled overall performance of a recursive blocked implementation using different optimized kernels.

From Figure 1, we see that with a GEMM performance of 500 Mflops/s and the slowest kernel (0.3 Mflops/s), the overall performance is at most approaching 50 % of the GEMM performance for $N = 2500$. But with the same GEMM performance and the faster kernels (executing at 3 and 10 Mflops/s, respectively), the overall performance rather quickly approaches 80–90 % of the performance of the DGEMM routine. This is of course, the performance behavior we would like to see in practice.

For $N = 250$, the number of flops performed in the kernel is 0.8 % of the total number of flops, but with the faster DGEMM and the slowest kernel, the time spent in the kernel is more than 93 % of the total time. With the same prerequisites and $N = 750$, the number of flops done in the kernel is less than 0.3 % of the total number of flops, while the time spent in the kernel is more than 80 % of the total time.

We remark that similar analyses hold for all matrix equations considered. For the IBM Power3, 200 MHz, the 0.3 Mflops/s value is roughly the average performance of the LAPACK DTGSY2 routine [32, 1] for these very small problems. DTGSY2 is the routine for solving small-sized generalized coupled Sylvester equations (3) used in an explicitly blocked level 3 solver [32]. The main reason for the poor performance is that the kernel algorithms in LAPACK are designed with the primary goal of producing high-accuracy results and signaling ill-conditioning. For example, the DTGSY2 solver is based on complete pivoting and overflow guarding. There is a trade-off between robustness, reliability and speed. In the next section, we discuss ways of improving the speed in the kernel design, without sacrificing the accuracy and robustness demands, at least as long as the problems are not too ill-conditioned.

5.2 Design of optimized superscalar kernels

The kernel routines in state-of-the-art libraries as LAPACK [1] and SLICOT [46] are typically not optimized for superscalar RISC processors. We continue our illustration with the LAPACK DTGSY2 kernel routine discussed above.

It uses DGETC2 to factorize the Z-matrix (see Z_{GCSY} in Section 2), and DGESC2 to do the forward and backward substitutions with overflow guarding. Several level 1, 2, and 3 BLAS routines (DAXPY, DSCAL, DGER, DGEMV, and DGEMM) are used in different updates. Now, DGETC2 and DGESC2 in turn use these and other BLAS routines to do the factorization and solving. While this is good programming practice and leads to good code reuse, the overhead of the routine calls is devastating to the performance. Typically, the compiler cannot make good intraprocedural optimizations, since the BLAS kernels are linked from a library and cannot be candidates for inlining. Besides, a large part of time spent in the BLAS routines is spent in setup code, i.e., parameter checking and collection of machine characteristics, which in turn degrade the performance.

Our approach is to design one single routine, which does all (or most of) the computations for solving the kernel problems using the Kronecker product matrix representation introduced in Section 2.2. The construction of the Z-matrix, the LU factorization using partial pivoting with overflow guarding, and the forward and backward substitutions are all done in the same routine. This leads to a great potential for register reuse. Also, by complete loop unrolling, the routine can make much better use of the superscalar characteristics of today's processors.

In Figure 1, it is assumed that the DGEMM performance is constant. This may look as a large simplification, but it is not. By the use of proper blocking [29, 30, 15] and optimization techniques for superscalar processors [15], it is possible to obtain high GEMM-performance already for quite small problems. For example, one of the DGEMM implementations which we have used for our results in Section 6, the IBM ESSL DGEMM routine, gives good performance also for small problem sizes ($40 < M, N, K < 200$). However, for even smaller problems, the overhead of the parameter checking and similar "wrapping" code degrade the performance. Therefore, we have developed a matrix-multiply kernel, without any parameter checking or cache blocking. Instead, it is implemented using register blocking techniques such as complete outer-loop unrolling and fusion. By doing so we facilitate for the compiler to look ahead and schedule the resulting assembly instructions optimally.

5.3 Choice of block size and ending of the recursion

The choice of block sizes is an important issue in algorithms which use explicit blocking. For a multi-level explicitly blocked algorithm, each block size must be chosen to match a specific level in the memory hierarchy, e.g., registers, level 1 or level 2 caches. This requires a deep knowledge of the architecture characteristics. On the other hand, for recursive blocked algorithms, we automatically obtain an hierarchical blocking which is variable and "squarish", and there is only need for one blocking parameter, *blks*. Typically, *blks* is chosen so that a few blocks of current sub-problems fit in the level 1 cache. For "problems" smaller than this size, recursion will not lead to any further speedup. Instead, to complete recursion until element level would typically cause too much overhead and a drop in performance (e.g., see [15, 16]).

For the **rtr*** solvers, the *blks* parameter can be chosen smaller than the size controlled by the level 1 cache without degrading the performance. This is due to the fact that there is enough computations at the leafs to hide the overhead caused by recursion. Although the superscalar kernel routines discussed in Section 5.2 are much faster than corresponding implementations in LAPACK and SLICOT, they are still much slower than the practical peak performance provided by an optimized GEMM kernel. The superscalar kernel solvers operate on problems of the size $2 \leq M, N \leq 4$, and this controls when the recursion will be ended. So, we use recursion also in the **tr*** block solvers now calling our superscalar GEMM kernel for all one-sided matrix-matrix updates. By doing so we minimize unnecessary overhead including costs for any buffer setups.

5.4 On the choice of BLAS implementation

We have shown that the recursive blocked algorithms reveal a great richness in matrix-matrix multiplication operations (GEMM). As the performance of the superscalar kernel increases, the GEMM routine plays a more important role for the overall performance (see Section 5.1).

The memory access pattern of the recursive blocked algorithms suggests that a BLAS implementation that uses recursion would perform very well together with the algorithms described here. However, this depends on the architecture characteristics. Recursive BLAS implementations have proven to be successful on processors with a large gap between the memory bandwidth and the computational power, e.g., see Gustavson et al. [14]. However, for machines with a balanced performance with respect to memory bandwidth and compute power, the difference is negligible, when the non-recursive BLAS is carefully tuned [38].

In [14], recursive blocked data formats that match recursive blocked algorithms are introduced and used in level 3 BLAS routines. Our experience is that using a recursive blocked data layout may lead to a significant performance gain, but it is not appropriate for our recursive triangular Sylvester-type matrix equation solvers in general. The main reason is the properties of the quasi-triangular matrices involved, where the 2×2 bulges along the block diagonal correspond to conjugate pairs of complex eigenvalues. A 2×2 diagonal block may force the recursive splitting to occur one row or column above or below the partitioning determined by the blocked format. This leads to spill rows and columns which increase the data management overhead. Therefore, we have abandoned its use. However, if all eigenvalues are real or we work in complex arithmetic all matrices will be upper triangular and a recursive blocked data format is straightforward to apply.

5.5 Parallelization issues

In our SMP implementations, different types of shared memory parallelism have been investigated. The first, and by far the easiest to implement, is to simply use level 3 BLAS routines that make use of more than one processor. Moreover, we can parallelize over independent tasks in the "Case 3" branch of the recursion of the algorithms, which include both GEMM updates and matrix equation solves from successive splittings (see **rtrsyct** in Section 3.1 and **rtrgcsy** in Section 3.3). Another way to go is to explicitly parallelize a standard blocked algorithm (e.g., see [31, 45, 44]). In this section, these three ways of parallelization including hybrid variants are discussed using the coupled Sylvester equation (3) and the algorithm **rtrgcsy** for illustration (see Section 3.3).

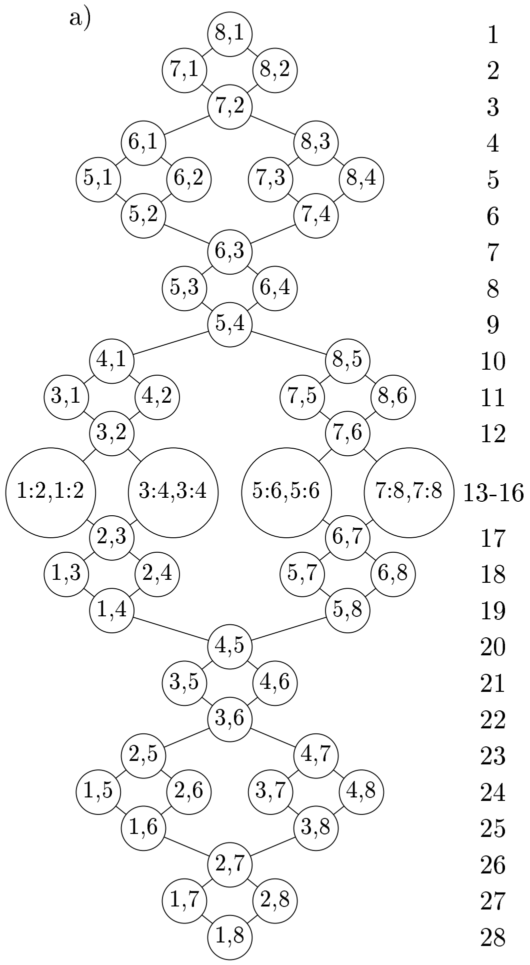
Implicit data parallelization. By linking to an SMP-aware implementation of BLAS, only the parallelism in the GEMM-based updates is taken into account. This gives fairly good speedup for SMP systems with a few number of processors, and, especially, for large problems, where the updates stand for a larger fraction of the total time to solve the problem. The main advantage of this approach is that it does not require any extra implementation work, granted that an SMP version of DGEMM is available. Moreover, this means that there are no requirements on the compiler or the run-time library to support any parallelization directives or routines, such as OpenMP or pthreads [42, 41].

Task parallelism in the recursion tree. In the **rtrgcsy** algorithm, there is also the possibility of parallelizing calls to the coupled Sylvester block kernel routine, by solving for the block matrix pairs $[X_{11}, Y_{11}]$ and $[X_{22}, Y_{22}]$ concurrently. Also, some of the updates are independent tasks which can be done in parallel. In our implementation of the **rtrgcsy** algorithm, we make us of an additional parameter *procs*, which holds the number of processors available to solve the current problem. If $procs > 1$, the problem size is large enough, and M, N fit into the Case 3 clause, then the second and the third recursive calls (solving for $[X_{11}, Y_{11}]$ and $[X_{22}, Y_{22}]$) are done in parallel, as well as some of the GEMM updates. The constraint on the problem size is necessary to prevent superfluous parallelization i.e., the cost of parallelizing the task (parallelization overhead) dominates the gain from executing it in parallel. This gives a diamond shaped execution graph of the kernel calls, see Figure 2. Each small circle in the graph corresponds to a task that should not be split any further.

In our implementation, the same SMP implementation of BLAS as above is used. While this is good for the largest updates at the top of the recursion, where the level of task parallelism is small, it is not optimal when there are many smaller updates to be done in parallel. Preferably, a DGEMM implementation where the number of available processors could be specified should be used. By using a DGEMM with a *proc* parameter, updates would alternate between making use of task parallelism (at the leafs of the recursion) and data parallelism (at the top-most roots of the recursion). However, we have not seen that option in today's BLAS implementations. Instead, with the current implementations there is a potential risk for over-parallelization; e.g., on a four-processor SMP, there could be four instances of DGEMM running in parallel, each trying to make use of four processors.

Another topic that leads to inconveniences with some compilers is the implementation of task parallelism. Our first choice is to use OpenMP for creating parallel sections. When more than two processors are used, the OpenMP PARALLEL SECTION directives are nested, each directive splits the current execution thread into two threads. While this code conforms to the OpenMP standard, the standard gives no guarantee that the inner PARALLEL SECTIONS will in fact be executed in parallel. For example, this is the case with the SGI MIPSpro 7 Fortran 90 compiler. For this and some other compilers, OpenMP is not a feasible option. Instead, pthreads can be used. One way is to create parallel tasks in the same fashion as in OpenMP. However, this technique has most likely larger overhead. Moreover, the implementation becomes complex due to pthreads' procedure based parallelism, especially if the updates are to be parallelized as well. Another way of using pthreads is to spawn all the threads, the same number of threads as the number of processors, in the beginning, and then letting all threads call the solver routine. Conditional expressions are used to control which data each thread is working on, and barriers are inserted in the code to keep threads synchronized. In the example in Figure 2, four threads (= the number of processors) enter from above, but only one thread is used to solve for the block $(8, 1)$. The other three threads are idle waiting until the task is completed. Not until time step 5, all four threads are involved in solving subsystems concurrently. But they are of course active in GEMM updates earlier. Both pthreads' based implementations are more complex, and they still give very poor performance with the MIPSpro compiler, since the SGI SCSL implementation of SMP BLAS is not compatible to pthreads.

In our reference implementation, the number of processors must be a power of two. However, this is



b)

15	16	18	19	24	25	27	28
13	14	17	18	23	24	26	27
11	12	15	16	21	22	24	25
10	11	13	14	20	21	23	24
5	6	8	9	15	16	18	19
4	5	7	8	13	14	17	18
2	3	5	6	11	12	15	16
1	2	4	5	10	11	13	14

c)

12	14	16	18	19	20	21	22
10	12	14	16	17	19	20	21
8	10	12	14	15	17	19	20
6	8	10	12	13	15	17	19
4	5	7	9	11	13	15	17
3	4	5	7	9	11	13	15
2	3	4	5	7	9	11	13
1	2	3	4	5	7	9	11

Figure 2: a) Execution graph of the **tr*** sub-system solver calls when solving a recursive blocked 8×8 problem on 4 processors. The numbers in the circles refer to the block indices in C/X and F/Y. The numbers to the right of the graph are time steps; e.g., in step 3, block $(7, 2)$ in X and Y are solved for. b) Time steps for every block in the previous problem. c) Time steps for every block, when solving an 8×8 blocked problem on 4 processors using the explicitly blocked SMP method.

not a general requirement of the algorithm. By not splitting the dimensions in even halves, the size of the different branches can be adjusted to match an uneven division of the processors.

Parallelization of explicitly blocked algorithms. Distributed memory as well as shared memory blocked algorithms for solving Sylvester-type matrix equations have been studied in the literature, e.g., see [31, 45, 44]. The above-cited algorithms try to utilize the maximum inherent parallelism of explicitly blocked algorithms for solving triangular matrix equations. For example, all blocks along each block diagonal of the solution(s) correspond to independent tasks (smaller triangular matrix equation solves) that can be executed in parallel.

A shared memory implementation of the triangular coupled Sylvester equation (3) is presented and discussed by Poromaa in [45, 44]. The solution (X, Y) overwrites the right hand sides (C, F), and the SMP algorithm solves for all blocks in each block diagonal of C and F in parallel, assuming enough number of processors are available. See Figure 2c for the execution order of the coupled Sylvester sub-system solves of a blocked 8×8 problem executing on four processors. We have implemented this algorithm with different block (sub-system) solvers, including the sequential **rtrgcsy** algorithm. The explicitly blocked algorithm also requires thread support from the compiler or a run-time library. However, if OpenMP is used, there are no complicated nested directives as in the recursive blocked algorithms. One advantage of this algorithm is the potential inherent parallelism in solving sub-systems along the block diagonals. One disadvantage is that the algorithm does not automatically generate good, squarish, calls to DGEMM. As such, the choice of block size is more architecture dependent than in a recursive blocked algorithm.

A remark on parallel tasks and sub-system solves. Assume that we have a square problem, and that all matrices are partitioned in $2^i \times 2^i$ equal-sized square blocks. Then all 4^i sub-systems to be solved are of the same size, and we assume that solving one such system takes one time step. For the recursive blocked algorithm with data parallelism only, the Sylvester sub-systems are solved in 4^i time steps, regardless the number of processors, and not counting the time of the updates. For the second recursive blocked algorithm which implements both task and data parallelism, the number of time steps for solving the 2^{2i} sub-systems on $p = 2^j$ processors is

$$K(i,j) = 2^{2i-j} + 2^{i-j} \sum_{u=0}^{j} \frac{(2^{j-u} - 1) \prod_{v=0}^{u-1} (i - v)}{u!}.$$

In the "toy example" illustrated in Figure 2, $i = 3, j = 2$, and $K(3, 2) = 28$. For the maximum of $p = 2^i$ processors, the algorithm requires a minimum of 3^i time steps for solving all Sylvester sub-systems. The third algorithm requires a minimum $2^i - 1$ time steps, provided there are enough processors. For the example in Figure 2, we only have four processors and therefore 22 time steps are required until the last sub-system $(1, 8)$ is solved. For larger, more realistic problems where $i \gg j$, both the second and the third algorithms solve the sub-systems in approximately $4^i/2^j$ time steps.

We remark that the discussion above only concerns the number of time steps for solving the sub-systems. The updates typically stand for a much larger fraction of the total work. For performance results and further discussions we refer to the next section.

6 Performance Results of Recursive Blocked Algorithms

The recursive blocked algorithms have been implemented in Fortran 90, using the facilities for recursive calls of subprograms, dynamic memory allocation and threads. In this section, we present sample performance results of these implementations executing on IBM Power, MIPS and Intel Pentium processor-based systems. We have selected a few processors representing different vendors and different architecture characteristics and memory hierarchies. The performance results (measured in Mflops/s) are computed using the flopcounts presented in Table 1 of Section 3.4. 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 libraries LAPACK [1] and SLICOT [46]. 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 LAPACK and SLICOT routines. For benchmark problems see [36, 37, 46].

Machine characteristics and BLAS implementations. The tests were run on four different machines. The first is the IBM RS/6000 SP SMP Thin Node, with four IBM PowerPC 604e CPUs running at 332 MHz. Each processor has a theoretical peak rate of 664 Mflops/s, a level 1 data cache of 32 kB, a level 2 cache of 256 kB, and a memory bandwidth of 1.3 Gb/s. The second is the IBM RS/6000 SP SMP Thin Node, with two IBM POWER3 CPUs running at 200 MHz. Each processor has a peak rate of 800 Mflops/s, a level 1 data cache of 64 kB, a level 2 cache of 4 MB. The memory bandwidth is 12.8 Gb/s. The operating system is AIX 4.3, and the compiler used is IBM XL Fortran 7.1. For BLAS, IBM ESSL 3.1.2 for SMP Architecture is used.

The third machine is one node of the HPC2N Linux cluster, with dual Intel Pentium III CPUs running at 550 MHz. Each processor has a theoretical peak rate of 550 Mflops/s, a level 1 data cache of 16 kB, a level 2 cache of 512 kB. The operating system is Linux 2.2.16, and the compiler used is Portland Group's pgf90 3.2. For BLAS, we have used ATLAS 3.2.1 with thread support, compiled with egcs 2.91.66. The fourth is one SGI Onyx2 node, with a MIPS R10000 CPU running at 195 MHz. The processor has a theoretical peak rate of 390 Mflops/s, a level 1 data cache of 32 kB, a level 2 cache of 4 MB. The operating system is IRIX 6.5, and the compiler used is MIPSpro 7 Fortran 90. For BLAS, the SGI Scientific Library 1.3 is used.

The level 3 BLAS operations, SYR2K in **rtrlyct** and TRMM in **rtrmfcn**, are called with quasi-triangular operands. Our implementations use the standard SYR2K and TRMM routines, which assume triangular operands, and some additional code to take care of any subdiagonal elements. The fixup code consists of a number of calls to the DSCAL routine, one call for every non-zero element in the subdiagonal.

6.1 Standard triangular Sylvester equation

In Figure 3, we show performance graphs for different algorithms and implementations, executing on IBM PowerPC 604e and Intel Pentium III processor-based systems, for solving triangular Sylvester equations. In Table 2, the measured performance and associated speedup results are presented.

The LAPACK DTRSYL implements an explicit Bartels-Stewart solver and is mainly a level 2 routine, which explains its poor performance behavior. Our recursive blocked **rtrsyct** shows between a two-fold and a 35-fold speedup with respect to LAPACK DTRSYL and an additional speedup up to 2.8 on a four processor PowerPC 604e node for large enough problems. The corresponding results on a two-processor Intel Pentium III show up to a 7-fold speedup with respect to LAPACK DTRSYL and an additional 1.6 speedup on two processors. The difference in the speedup B/A for the two architectures is mainly do to lower cache-miss penalties for the Pentium processor, which in turn leads to less degradation in the performance of LAPACK DTRSYL. The strong dependencies in the dataflow execution graph limit the possible parallel speedup for the triangular Sylvester equation.

	IBM PowerPC 604e					Intel Pentium III				
	Mflops/s		Speedup			Mflops/s		Speedup		
$M = N$	A	B	B/A	C/B	D/B	A	B	B/A	C/B	D/B
100	75.2	184.7	2.45	0.94	0.93	133.9	208.2	1.55	0.99	1.22
250	34.1	263.7	7.74	1.60	1.91	80.2	268.9	3.35	1.11	1.43
500	14.8	296.8	20.01	1.97	2.13	51.5	309.0	6.00	1.25	1.43
1000	9.5	332.2	35.11	2.42	2.37	47.3	343.6	7.26	1.41	1.52
1500	-	342.4	-	2.62	2.51	-	352.5	-	1.51	1.56
2000	-	363.3	-	2.81	2.52	-	366.0	-	1.58	1.60

A – LAPACK DTRSYL
B – **rtrsyct** with new kernel solver
C – B + linking with SMP BLAS
D – C + utilizing explicit // in the recursion tree

Table 2: Performance results for the triangular Sylvester equation—IBM PowerPC 604e (left) and Intel Pentium III (right). Labels A–D represent different algorithms and implementations.

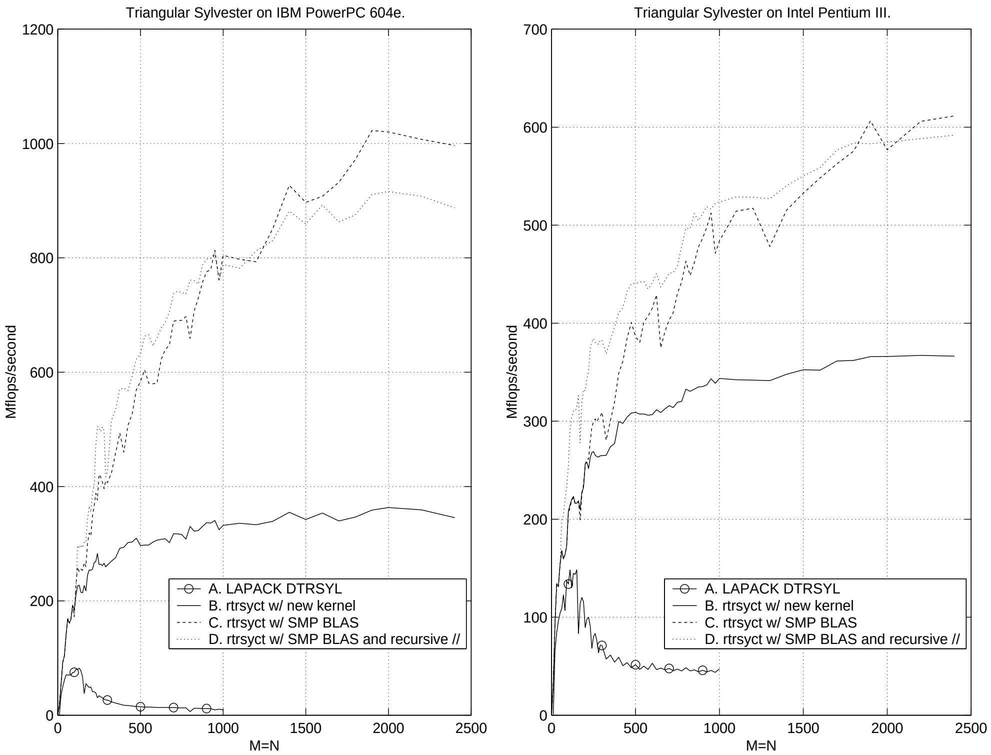


Figure 3: Performance results for the triangular Sylvester equation ($M = N$)—IBM SP PowerPC 604e (left) and Intel Pentium III (right).

6.2 Triangular Lyapunov equation

In Figure 4 and Table 3, performance results for the triangular Lyapunov equation are presented, now using IBM Power3 and SGI MIPS R10000 processor-based systems. Since there is no Lyapunov solver in LAPACK, we compare our recursive blocked **rtrlyct** algorithm with the SLICOT SB03MY 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 triangular Sylvester equation. We remark that the speedup of **rtrlyct** with respect to the SLICOT SB03MY is between 2 and 83 and an additional speedup of 1.7 on a two-processor IBM Power 3 node for large enough problems. The results for the MIPS R10000 show between an 1.5-fold to over 25-fold speedup. Due to lack of support for nested multithreading 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).

	IBM Power3				MIPS R10000		
	Mflops/s		Speedup		Mflops/s		Speedup
N	A	B	B/A	C/B	A	B	B/A
100	77.0	166.5	2.16	0.89	82.0	123.5	1.51
250	85.3	344.5	4.04	1.11	88.7	224.5	2.53
500	10.6	465.0	43.85	1.33	42.2	277.8	6.58
1000	7.7	554.7	72.20	1.56	14.5	254.0	17.57
1500	7.0	580.5	83.19	1.64	9.7	251.0	25.81
2000	-	611.1	-	1.73	-	246.0	-

A – SLICOT SB03MY
B – **rtrlyct** with new kernel solver
C – B + linking with SMP BLAS

Table 3: Performance results for the triangular Lyapunov equation—IBM Power3 (left) and SGI Onyx2 MIPS R10000 (right). Labels A–C represent different algorithms and implementations.

6.3 Triangular coupled Sylvester equation

In Figures 5 and 6, performance graphs for different algorithms and implementations, executing on IBM PowerPC 604e and Intel Pentium III processor-based systems, for solving triangular generalized coupled Sylvester equations are displayed. In Table 4, we present the measured performance and associated speedup results.

In total, we make comparisons between seven different implementations. Two of them are the LAPACK DTGSYL (A) and a parallel variant which uses SMP BLAS (F). We remind that LAPACK DTGSYL is an explicitly blocked level 3 algorithm based on a generalization of the Bartels-Stewart algorithm [32, 35]. Three of them are the sequential recursive blocked **rtrgcsy** (B) and two parallel variants (C and D). The last two (E and G) are new explicitly parallelized implementations of the blocked method [32, 44, 45].

Since the LAPACK DTGSYL is mainly a level 3 routine, its performance increases with increasing problem sizes and levels out due to only one level of blocking. But still **rtrgcsy** shows over a five-fold speedup with respect to LAPACK DTGSYL and an additional speedup up to 3.2 on a four processor PowerPC 604e node for large enough problems. The corresponding results on a two-processor Intel Pentium III are up to 2.6-fold speedup with respect to LAPACK DTRSYL and additionally up to an 1.6-fold speedup on two processors. In contrary to the results for the triangular Sylvester and Lyapunov equations, the relative performance of the LAPACK routine with respect to the recursive blocked algorithm (column B/A in Table 4) is decreasing with increasing problem sizes. At first this looks counter-intuitively, but the results verify the impact of a fast kernel solver in the level 3 algorithms, especially for small to medium-sized problems (see Section 5.1).

The good results for the explicitly blocked parallel implementation using our new superscalar kernel solver (E) also verifies the kernel impact on the parallel performance (see columns E/B in Table 4).

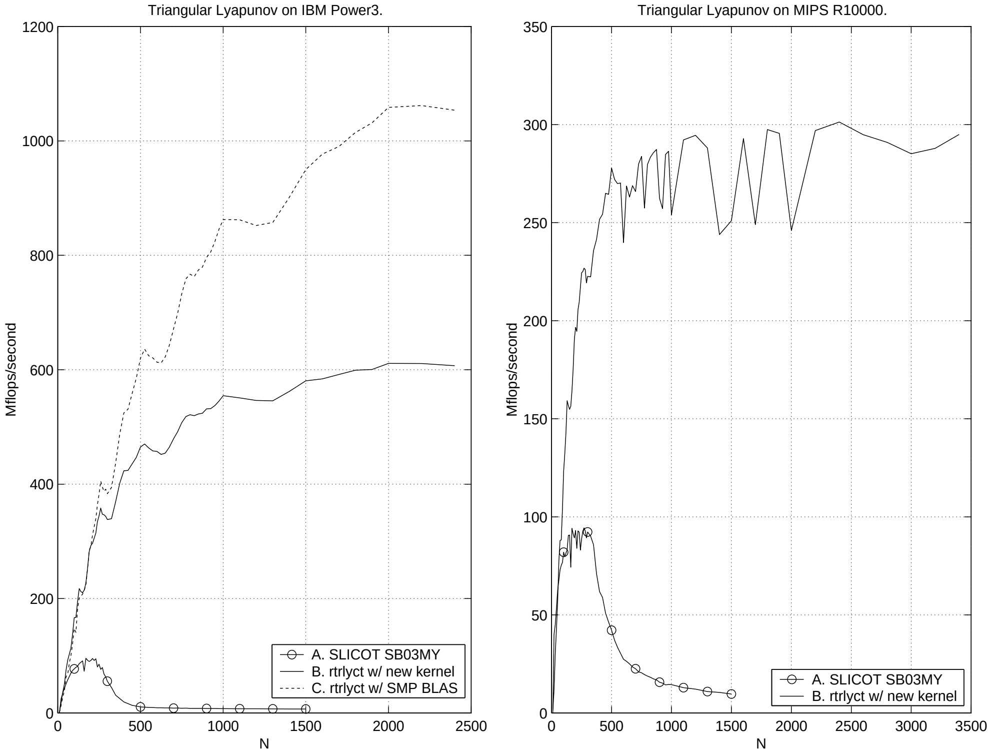


Figure 4: Performance results for the triangular Lyapunov equation—IBM Power 3 (left) and SGI Onyx2 MIPS R1000 (right).

Similar results for the explicitly blocked parallel implementation using the LAPACK DTGSY2 kernel (G) are much worse (see column G/A for the Intel processor in Table 4).

6.4 Impact on solving a unreduced matrix equation

We have also investigated the impact of the choice of triangular matrix equation solver on the time to solve an unreduced matrix equation. The reduction of an unreduced matrix equation to a triangular counterpart and the backtransformation of the solution are operations which both are at least as costly (measured in flops) as the triangular solve. Nevertheless, using our recursive blocked solvers can give a remarkable performance improvement. We use the SB03MD routine in the SLICOT library [46] for illustration, which solves unreduced continuous-time Lyapunov equations. SB03MD also has an option to compute an estimate of the separation Sep[LYCT] between A and $-A^T$.

In Table 5a, timings for the SB03MD routine are displayed for problem sizes ranging from 50 to 1000 using two different triangular matrix equation solvers. These solvers are SB03MY provided in SLICOT, which implements a standard Bartels-Stewart method calling BLAS, and our recursive blocked **rtrlyct** algorithm. In the second column, the total times for solving the unreduced system with SB03MY as the triangular solver are displayed. This includes the time for the Schur factorization and backtransformation of the solution. In the fourth and fifth columns, similar results are displayed when SB03MY is replaced by the **rtrlyct** routine. We see between 75 % to 100 % speedup for problem sizes $N \geq 500$. We remark that the big difference of the timings for $N = 250$ and $N = 500$ and their impact are due to cache effects, which automatically are taken care of by our recursive blocked algorithms.

	IBM PowerPC 604e						
	Mflops/s		Speedup				
$M = N$	A	B	B/A	C/B	D/B	E/B	F/A
100	33.9	180.2	5.31	0.88	1.23	0.95	0.73
250	67.6	254.5	3.77	1.60	2.09	1.43	0.81
500	98.7	286.2	2.90	2.03	2.78	2.19	0.93
1000	138.2	325.5	2.36	2.36	2.93	2.97	1.16
1500	179.8	336.0	1.87	2.56	3.19	3.16	1.27
2000	201.7	358.9	1.78	2.75	3.11	3.24	1.41
	IBM PowerPC 604e						
M	Mflops/s		Speedup				
$(N = \frac{M}{10})$	A	B	B/A	C/B	D/B	E/B	F/A
100	20.1	88.1	4.38	0.61	0.96	0.91	0.71
250	44.1	184.6	4.18	1.23	1.25	1.18	0.75
500	71.3	246.7	3.46	1.66	1.65	1.56	0.84
1000	112.8	295.2	2.62	2.01	2.31	1.87	0.97
1500	141.6	293.3	2.07	2.39	2.72	2.34	1.07
2000	162.5	324.0	1.99	2.39	2.90	2.78	1.21
	Intel Pentium III						
	Mflops/s		Speedup				
$M = N$	A	B	B/A	C/B	D/B	G/A	
100	48.8	103.6	2.12	1.70	2.17	0.98	
250	95.8	250.8	2.62	1.11	1.39	1.07	
500	141.2	298.0	2.11	1.23	1.39	1.22	
1000	194.7	335.5	1.72	1.40	1.53	1.29	
1500	221.7	344.7	1.55	1.51	1.59	1.21	
2000	238.0	362.1	1.52	1.57	1.59	1.26	

A – LAPACK DTGSYL (with DTGSY2 kernel)
B – **rtrgcsy** with new kernel solver
C – B + linking with SMP BLAS
D – C + utilizing explicit // in the recursion tree
E – Explicit // using SMP BLAS and new kernel
F – LAPACK DTGSYL linking with SMP BLAS
G – Explicit // using SMP BLAS and LAPACK kernel

Table 4: Performance results for the triangular generalized coupled Sylvester equation—IBM PowerPC 604e (top, middle) and Intel Pentium III (bottom). Labels A–G represent different algorithms and implementations.

In Table 5b, we present timings for both solving an unreduced Lyapunov equation and computing a 1-norm-based-estimate of the separation Sep[LYCT] between A and $-A^T$ (see Section 2.2). The condition estimation process leads to several calls to the triangular Lyapunov solver. Here, the impact is remarkable, with a four to five-fold speedup for problem sizes $N \geq 500$.

7 Conclusions and future work

The performance results verify that our recursive approach is very efficient for solving one-sided triangular Sylvester-type matrix equations on today's hierarchical memory computer systems. 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. To complete recursion until element level

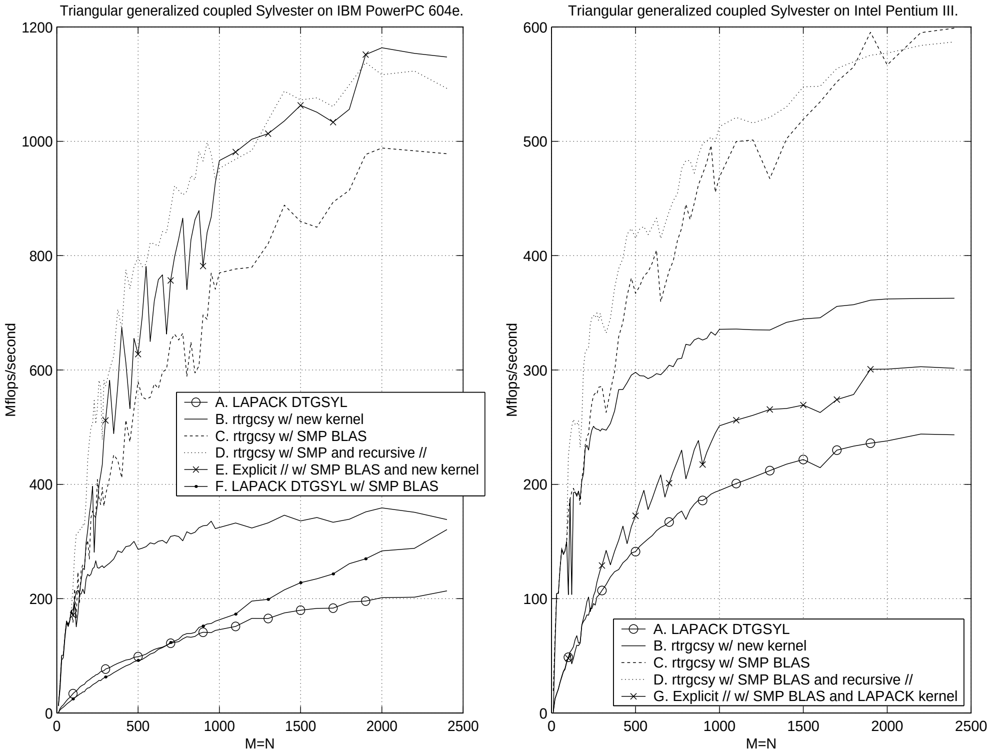


Figure 5: Performance results for the generalized coupled Sylvester equation ($M = N$)–IBM SP PowerPC 604e (left) and Intel Pentium III (right).

a)	SB03MD using SB03MY		SB03MD using **rtrlyct**		
N	Total time	Solver part	Total time	Solver part	Speedup
50	0.0253	10.5 %	0.0238	8.8 %	1.06
100	0.141	9.2 %	0.134	4.7 %	1.05
250	1.65	11.1 %	1.52	3.1 %	1.09
500	26.8	43.7 %	15.3	1.9 %	1.75
750	105.8	48.5 %	54.4	1.6 %	1.94
1000	258.4	50.0 %	131.5	1.4 %	1.97

b)	SB03MD using SB03MY		SB03MD using **rtrlyct**		
N	Total time	Solver part	Total time	Solver part	Speedup
50	0.0358	34.7 %	0.0338	26.0 %	1.06
100	0.205	34.5 %	0.164	18.5 %	1.25
250	2.37	35.7 %	1.74	13.0 %	1.36
500	64.9	76.8 %	16.5	8.4 %	3.94
750	278.2	80.6 %	59.1	8.7 %	4.71
1000	706.4	81.6 %	141.4	7.9 %	4.99

Table 5: In a), the total execution time for solving a unreduced Lyapunov equation is displayed for two different triangular Lyapunov equation solvers. In b), an estimate of the separation Sep[LYCT] between A and $-A^T$ is also computed.

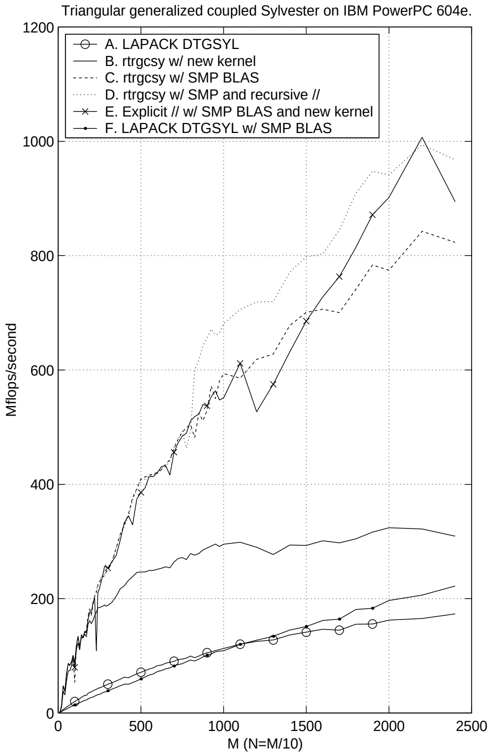


Figure 6: Performance results for the generalized coupled Sylvester equation ($N = M/10$)–IBM SP PowerPC 604e.

would in general cause too much overhead and a drop in performance. Our solution is to develop new high-performance superscalar kernels for small-sized triangular Sylvester-type 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. Altogether, this leads to much faster algorithms for solving reduced as well as unreduced Sylvester-type matrix equations and different associated condition estimation problems.

Our future work is focused on the development of recursive blocked algorithms for *two-sided* matrix equations, which include matrix product terms of type $\mathrm{op}(A)\mathrm{op}(X)\mathrm{op}(B)$, where $\mathrm{op}(Y)$ can be Y or its transpose, Y^T. Some examples are the discrete-time standard and generalized Sylvester and Lyapunov equations [3, 11, 18, 4, 35, 9, 32, 43, 37]. Results will be presented in a Part II paper [23].

Acknowledgements

This research was conducted using the resources of the High Performance Computing Center North (HPC2N) and PDC–Parallelldatorcentrum at KTH, Stockholm. During this work resources at the Danish Computing Centre for Research and Education (UNI•C), Lyngby, Denmark have also been used. We also thank Fred Gustavson and our recursive pals 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] E. Elmroth and F. Gustavson. High-Performance Library Software for QR Factorization, in *Applied Parallel Computing: New Paradigms for HPC Industry and Academia*, Lecture Notes in Computer Science, vol. 1947, pp 53–63, Springer Verlag, 2001.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

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

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

[22] 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.

[23] I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations, Report UMINF 01.06, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, 2001 (to appear).

[24] B. Kågström. Bounds and perturbation bounds for the matrix exponential, *BIT*, (17):39–57, 1977.

[25] B. Kågström. Numerical computation of matrix functions, Report UMINF-58.77, Institute of Information Processing, Umeå University, S-901 87 Umeå, Sweden, 1977.

[26] B. Kågström. A Direct Method for Reordering Eigenvalues in the Generalized Real Schur Form of a Regular Matrix Pair (A, B), in *Linear Algebra for Large Scale and Real–Time Applications*, M. Moonen, G. Golub, and B. D. Moor, eds., Kluwer Academic Publishers, Amsterdam, 1993, pp. 195–218.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] B. Kågström and P. Poromaa. Computing Eigenspaces with Specified Eigenvalues of a Regular Matrix Pair (A, B) and Condition Estimation, *Numerical Algorithms*, 12:369–407, 1996.

[34] B. Kågström and P. Van Dooren. A generalized state-space approach for the additive decomposition of a transfer matrix, *Int. J. Numerical Linear Algebra with Applications*, (1):165–181, 1992.

[35] 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.

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

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

[38] 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.

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

[40] C. B. Moler and C.F. Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix, *SIAM Rev.*, 20:801–836, 1978.

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

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

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

[44] 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.

[45] 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.

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

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

[48] C.F. Van Loan. The Sensitivity of the Matrix Exponential, *SIAM J. Num. Anal.*, 14:971–981, 1977.