Formal Derivation of Algorithms: The Triangular Sylvester Equation

Enrique S. Quintana-Ortí* Robert A. van de Geijn†

September 18, 2001

Abstract

In this paper we apply a formal approach for the derivation of dense linear algebra algorithms to the triangular Sylvester equation. The result is a large family of provably correct algorithms. By using a coding style that reflects the algorithms as they are naturally presented, the correctness of the algorithms carries through to the correctness of the implementations. Analytically motivated heuristics are used to subsequently choose members from the family that can be expected to yield high performance. Finally, we report performance on the Intel (R) Pentium III processor that is superior to that reported previously in the literature for this operation.

1 Introduction

In a recent paper the Formal Linear Algebra Methods Environment (FLAME) was introduced [10]. FLAME is both a systematic approach for deriving (dense) linear algebra algorithms and a library for the implementation of the resulting algorithms. The rationale is that by formally deriving algorithms, correctness can be asserted. Moreover, by providing a framework for coding that mirrors the derived algorithms, the opportunity for the introduction of coding errors is greatly reduced and thus the correctness of the algorithms carries through to the implementations. In that paper the simple example of LU factorization was used to illustrate the basic techniques. In this paper, we

*Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.080–Castellón, Spain, quintana@icc.uji.es

†Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, rvdg@cs.utexas.edu

demonstrate the versatility of FLAME by concentrating on a more complex linear algebra operation, the solution of a triangular Sylvester equation.

While the solution of the triangular Sylvester equation is a well-studied problem, this paper presents a number of contributions:

- An illustration of the application of FLAME to a problem arising in control theory.
- The derivation of a large family of provably correct algorithms which includes, as a small subset, algorithms that are closely related to known traditional methods as well as recently proposed recursive algorithms.
- An analysis that provides heuristics for composing members of the family to yield the best performance.
- A demonstration of performance that is superior to any previously reported.

Altogether, dozens of new, high-performance, algorithms and implementations are given.

While this paper is written to be self-contained, it is highly recommended that the reader consults the earlier paper on FLAME as well as a recent paper that gives theoretical insight into high-performance matrix multiplication algorithms [9]. This paper is structured as follows: In Section 2, we review the triangular Sylvester equation and traditional algorithms for its solution. We derive algorithms that are closely related to traditional algorithms in Section 3, and a more general family in Section 4. In Section 5 we describe insights that we use to identify candidates from the family that are likely to yield the best performance. Performance results on an Intel (R) Pentium III processor are given in Section 6. Concluding remarks follow in the final section.

2 The triangular Sylvester equation

Consider the Sylvester equation

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

where A is an $m \times m$ matrix, B is $n \times n$, C and X are $m \times n$, and X is the sought-after solution. Let $\Lambda(A) = \{ \alpha_i \}_{i=1}^{m}$ and $\Lambda(B) = \{ \beta_j \}_{j=1}^{n}$ denote, respectively, the eigenspectra of A and B; then (1) has a (unique) solution if and only if $\alpha_i + \beta_j \neq 0$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. For further details on the existence of solutions of the Sylvester equation and numerical solvers see, e.g., [3, 7, 11].

Sylvester equations have numerous applications in control theory, signal processing, filtering, image restoration, the decoupling of ordinary and partial differential equations, and block-diagonalization of matrices; see, e.g., [1, 5, 8, 14]. Also note that $B = A^T$ yields the Lyapunov equation such that everything derived here can be used (and simplified) for this type of equations playing a vital role in many areas of computer-aided control system design.

Here we focus on the triangular case of the Sylvester equation where the coefficient matrices, A and B, are upper triangular. (Though any or both of these matrices could have been reduced instead to lower triangular form, the study of these other cases leads to algorithms similar to those described in this paper.) This triangular form is a "by-product" obtained, e.g., in the solver developed in [3]. In their algorithm, the coefficient matrices are first reduced to the Schur form using the QR algorithm [8], and the corresponding transformation matrices are applied to C. The triangular Sylvester equation thus obtained is solved using a back-substitution procedure similar to a triangular linear system with multiple right-hand sides. The cost of solving the triangular Sylvester equation of dimension $m \times n$, using a traditional serial (non-blocked) algorithm, is $m^2n + mn^2$ floating point operations [8]. Once this triangular equation is solved, the inverse transformations are applied to recover the solution of the original equation.

If all matrices in the equation have real entries and only real arithmetic is desired, the QR algorithm can be used to obtain the real Schur (or quasi-triangular) form of the coefficient matrices, a block upper triangular form with 1×1 or 2×2 diagonal blocks corresponding respectively to real eigenvalues or pairs of complex eigenvalues of the matrices.

Blocked algorithms usually obtain a higher performance in modern computers by rearranging the computations as possible in terms of matrix multiplication [6]. LAPACK [2] is a library that illustrates the benefits of reformulating algorithms to be rich in matrix-matrix products. Some of the latest research on high-performance implementation of matrix multiplication is embodied in the packages ATLAS [15], PHIPAC [4], and ITXGEMM [9].

Blocked algorithms for solving the triangular Sylvester equation can easily be derived from the serial algorithms and are usually composed of two nested loops which iterate over blocks of columns and rows of the solution matrix. For each iteration of the inner loop a new block of the solution is obtained. Depending on the algorithm, some updates may be needed before a new block of the solution is obtained (leading to a lazy algorithm, which postpones much of the work) or after it is computed (an eager algorithm in such case).

As an example, we next present a traditional row-lazy/column-eager blocked triangular Sylvester equation solver. Assume A is partitioned into $b_m \times b_m$ blocks, $A_{i,j}$, $i,j = 1,\ldots,m/b_m$, and B is partitioned into $b_n \times b_n$ blocks, $B_{i,j}$, $i,j = 1,\ldots,n/b_n$. Hereafter, we assume that m and n are integer multiples of b_m and b_n, respectively. These partitions induce conformal partitions of X and C into $b_m \times b_n$ blocks. Setting both b_m and b_n to 1 leads to element-wise algorithms, while setting only one of them produces row-oriented or column-oriented variants.

The algorithm is stated in Figure 1, where we borrow the colon notation from MATLAB. This algorithm can easily be modified to overwrite C with the solution of the equation. The Sylvester equation arising at each iteration of the inner loop is usually solved using a non-blocked, row-oriented or column-oriented version of the algorithm. Notice that, just before a new block of the solution is obtained, the corresponding block-row of C is updated with respect to the previous blocks of X in the same block-column, leading to a row-lazy updating scheme. On the other hand, when this new block is computed, it is used to update the remaining blocks of C in the same block-row in a column-eager updating scheme. Three more variants of the algorithm are obtained by rearranging the updates to be row-lazy/eager and column-lazy/eager [13].

$$
\begin{array}{l}
\texttt{for } i = m/b_m : -1 : 1 \\
\quad \texttt{for } j = 1 : n/b_n \\
\quad\quad C_{i,j} = C_{i,j} - A_{i,i+1:m/b_m} X_{i,i+1:m/b_m,j} \\
\quad\quad \texttt{solve } A_{i,i} X_{i,j} + X_{i,j} B_{j,j} = C_{i,j} \\
\quad\quad C_{i,j+1:n/n_b} = C_{i,j+1:n/n_b} - X_{i,j} B_{j,j+1:n/n_b} \\
\quad \texttt{end} \\
\texttt{end}
\end{array}
$$

Figure 1: Row-lazy/column-eager blocked triangular Sylvester equation solver.

Recursive variants of these solvers have been recently developed in [12]. Briefly, a recursive algorithm employs the same algorithm for solving the Sylvester equation in the inner loop, but uses a smaller dimension of the block sizes b_m and b_n. The higher efficiency of these algorithms is obtained by decoupling the dimensions of the blocks for the matrix multiplications from those of the Sylvester equations. The goal is to perform as much of the computation in terms of matrix multiplications as is possible, while maximizing the size of the matrices involved in these products.

3 Row- or Column-Oriented Algorithms

We first derive two block-row oriented (with respect to matrix C) solvers by partitioning only the first of the coefficient matrices, A (see Subsection 3.1). Analogous block-column oriented versions are obtained by partitioning B instead of A, as suggested in Subsection 3.2.

3.1 Block-row oriented solvers

Let us consider equation (1) where, in order to reduce the number of matrices, we want to overwrite matrix C with the solution of the equation, X.

We start our derivation of block-row oriented algorithms by partitioning matrix A into four quadrants

$$A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} = 0 & A_{BR} \end{array} \right),$$

where A_{BR} is a $k_m \times k_m$ block. The indices $\{TL\}$, $\{TR\}$, $\{BL\}$, and $\{BR\}$ stand for top-left, top-right, bottom-left, and bottom-right, respectively. Accordingly, we next apply a conformal partition to X and C by blocks of rows

$$X \rightarrow \left(\begin{array}{c} X_T \\ \hline\hline X_B \end{array} \right), \quad C \rightarrow \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right),$$

where X_B and C_B are $k_m \times n$ blocks. Here $\{T\}$ and $\{B\}$ stand for top and bottom, respectively.

With these partitionings, equation (1) can be rewritten as

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array} \right) \left(\begin{array}{c} X_T \\ \hline\hline X_B \end{array} \right) + \left(\begin{array}{c} X_T \\ \hline\hline X_B \end{array} \right) B = \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right),$$

and multiplying out the left-hand-side of the matrix equation, we obtain the following equalities

$$\begin{array}{rcl} A_{TL}X_T + A_{TR}X_B + X_T B & = & C_T, \\ A_{BR}X_B + X_B B & = & C_B. \end{array}$$

Thus, the computation of X requires solving two different Sylvester equations and performing a matrix update on C_T, which we designate as follows:

$$\begin{array}{rcl} \Omega(X_T) & \equiv & A_{TL}X_T + X_TB = C_T, \\ \Omega(X_B) & \equiv & A_{BR}X_B + X_BB = C_B, \\ \bar{C}_T & \equiv & C_T \leftarrow C_T - A_{TR}X_B. \end{array} \tag{2}$$

Notice that there are data dependencies which induce a strict order on the sequence of operations. First, the solution of $\Omega(X_B)$ is obtained, then the update $\bar{C}_T$ is computed and, finally, $\Omega(X_T)$ is solved.

The FLAME approach to deriving algorithms based on these partitionings starts by considering the following two questions:

i) What part of the computation has already been performed at a certain stage (iteration)? The answer is a condition that is called the *loop-invariant* for the algorithm.

ii) How can we advance the computation so that the loop-invariant is satisfied at the beginning of the next stage? The answer to this question yields the updates (to the matrix) that comprise the body of the loop.

To answer the first question we consider a certain (intermediate) situation (case) where some of the operations in (2) have been performed. The instances where no operations have been performed or all of them are already performed are not considered valid *intermediate* cases. With three different operations and the data dependencies, we have only two valid cases:

Case	Operations completed	Current contents of C
R1	$\Omega(X_B)$	$\left(\begin{array}{c} C_T \\ \hline\hline X_B \end{array}\right)$
R2	$\bar{C}_T$, $\Omega(X_B)$	$\left(\begin{array}{c} C_T - A_{TR}X_B \\ \hline\hline X_B \end{array}\right)$

The current content of C (right-most column) becomes the condition that defines the loop-invariant of the algorithm.

Now, in order to answer the second question, we derive the steps that allow the computation of X to proceed forward (up) by b_m rows, i.e., the operations that will allow the loop-invariant to be satisfied at the beginning of the next iteration. To derive these steps, we repartition matrix A as

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right),$$

where A_{11} is an $b_m \times b_m$ block. The parameter b_m determines the granularity of our block-row oriented algorithm. By setting $b_m = 1$ we obtain a non-blocked row-oriented algorithm. (If A is quasi-triangular, b_m can be varied during the computation so that a 2×2 diagonal block is not divided between two diagonal blocks in this last partition.) Conformally, apply the following repartitions to X and C

$$\left(\begin{array}{c} X_T \\ \hline\hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline\hline X_2 \end{array}\right), \quad \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array}\right),$$

where X_1 and C_1 are $b_m \times n$ blocks. The double lines in these partitionings indicate how far the computation has progressed.

3.1.1 Lazy algorithm

If we wish to maintain condition R1, we assume that X_2 is currently available, while C_2 and A_{22} have been used for this purpose. Moving the computation forward by b_m rows therefore is equivalent to moving the double lines up by b_m rows, so that X_1 is also available. In other words, we know that currently C holds

$$\left(\begin{array}{c} C_T \\ \hline\hline X_B \end{array}\right) = \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline X_2 \end{array}\right).$$

Now, consider that the computation has effectively moved forward by b_m rows, leading to

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} X_T \\ \hline\hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ \hline\hline X_1 \\ \hline X_2 \end{array}\right), \quad \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array}\right).$$

Notice that after the lines have moved, we need the contents of C to become

$$\left(\begin{array}{c} C_T \\ \hline\hline X_B \end{array}\right) = \left(\begin{array}{c} C_0 \\ \hline\hline X_1 \\ \hline X_2 \end{array}\right).$$

The question now is, for case R1, what is the sequence of operations which allows the computation to move forward while maintaining the indicated loop-invariant. As the Sylvester equation for X_2 was already solved at the beginning of the stage, we conclude that we need only to perform the operations:

$$
\begin{aligned}
C_1 &\leftarrow C_1 - A_{12} X_2, \\
A_{11} X_1 + X_1 B &= C_1.
\end{aligned}
$$

This procedure leads us to an algorithm that can be classified as "lazy". Before computing a new block-row of X, this block is updated with respect to the block-rows of X that have been previously computed. The algorithm is stated in Figure 2. To illustrate how the FLAME library allows code to mirror the derived algorithm, thus largely inheriting the proven correctness, an implementation using FLAME is given in Figure 4.

For those more comfortable with traditional algorithms, this is equivalent to the solver in Figure 3 (left). The lazy algorithms just presented, in the FLAME and the traditional formulations, are special cases of the algorithm in Figure 1, with $b_n = n$.

3.1.2 Eager algorithm

The eager variant of the algorithm is obtained from case R2, as we show next. In this case the computation has proceeded forward to solve $\Omega(X_B)$ and perform the update $\bar{C}_T$. Thus, C currently contains:

$$
\left(\begin{array}{c} C_T - A_{TR} X_B \\ \hline\hline X_B \end{array}\right) = \left(\begin{array}{c} C_0 - A_{02} X_2 \\ \hline C_1 - A_{12} X_2 \\ \hline\hline X_2 \end{array}\right).
$$

With the boundaries of the computation moved forward by b_m rows, the loop-invariant that must be satisfied for the next stage is given by

$$
\left(\begin{array}{c} C_T - A_{TR} X_B \\ \hline\hline X_B \end{array}\right) = \left(\begin{array}{c} C_0 - A_{02} X_2 - A_{01} X_1 \\ \hline\hline X_1 \\ \hline X_2 \end{array}\right)
$$

Thus, in order to move the computation forward while maintaining the loop-invariant for this case, we conclude that we need to perform the following operations:

$$
\begin{aligned}
A_{11} X_1 + X_1 B &= C_1, \\
C_0 &\leftarrow C_0 - A_{01} X_1.
\end{aligned}
$$

Algorithm 1 $C \leftarrow X$, *where* $AX + XB = C$
(Block-row oriented)

partition

$$A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), \quad C \rightarrow \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right),$$

where A_{BR} **is** 0×0 **and** C_B **is** $0 \times n$

do until C_B **is** $0 \times n$

determine block size b_m

repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right), \quad \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array}\right)$$

where A_{11} **is** $b_m \times b_m$ **and** C_1 **is** $b_m \times n$

Lazy variant:	*Eager variant:*
$C_1 \leftarrow C_1 - A_{12}X_2$	$C_1 \leftarrow X_1, where$
$C_1 \leftarrow X_1, where$	$A_{11}X_1 + X_1B = C_1$
$A_{11}X_1 + X_1B = C_1$	$C_0 \leftarrow C_0 - A_{01}X_1$

continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right), \quad \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \leftarrow \left(\begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array}\right)$$

enddo

Figure 2: Lazy and eager row-block oriented triangular Sylvester equation solvers derived from R1 (lazy) and R2 (eager).

`for` $i = m/b_m : -1 : 1$

Lazy variant:	***Eager variant:***
$C_{i,:} = C_{i,:} - A_{i,i+1:m/b_m}X_{i+1:m/b_m,:}$	`solve` $A_{i,i}X_{i,:} + X_{i,:}B = C_{i,:}$
`solve` $A_{i,i}X_{i,:} + X_{i,:}B = C_{i,:}$	$C_{1:i-1,:} = C_{1:i-1,:} - A_{1:i-1,i}X_{i,:}$

`end`

Figure 3: Lazy and eager traditional row-block oriented triangular Sylvester equation solvers.

```
int FLA_Syl_Lazy_Block-Row( FLA_Obj A, FLA_Obj B, FLA_Obj C, int bm )
{
  // Declaration of local objects...

  FLA_Part_2x2( A,   &ATL, /**/ &ATR,
                     /* ********** */
                     &ABL, /**/ &ABR,
          /* with */ 0, /* by */ 0, /* submatrix */ FLA_BR );

  FLA_Part_2x1( C, &CT,
                   /**/
                   &CB,
          /* with length */ 0, /* submatrix */ FLA_BOTTOM );

  while ( FLA_Obj_length( CT ) != 0 ){

    FLA_Repart_2x2_to_3x3( ATL, /**/ ATR,       &A00,  &A01,    /**/  &A02,
                                /**/            &A10,  &A11,    /**/  &A12,
                          /* ******** */        /* ********************* */
                          ABL, /**/ ABR,        &A20,  &A21,    /**/  &A22,
          /* with */ bm, /* by */ bm, /* A11 split from */ FLA_TL );

    FLA_Repart_2x1_to_3x1( CT,                  &C0,
                                                &C1,
                         /**/                   /**/
                          CB,                   &C2,
          /* with length */ bm, /* C1 split from */ FLA_TOP );

    /* **************************************************************** */

    /* C1 <- C1 - A12 X2 */
    FLA_Gemm( FLA_NO_TRANSPOSE, FLA_NO_TRANSPOSE,
              MINUS_ONE, A12, C2, ONE, C1 );

    /* C1 <- X1, where X1 solves A11 X1 + X1 B = C1 */
    FLA_Syl_level2( A11, B, C1 );

    /* ***************************************************************** */

    FLA_Cont_with_3x3_to_2x2( &ATL, /**/ &ATR,      A00, /**/ A01,      A02,
                             /* ********** */     /* ****************** */
                                   /**/             A10, /**/ A11,      A12,
                             &ABL, /**/ &ABR,       A20, /**/ A21,      A22,
          /* with A11 added to submatrix */ FLA_BR );

    FLA_Cont_with_3x1_to_2x1( &CT,                  C0,
                             /**/                  /**/
                                                    C1,
                             &CB,                   C2,
          /* with C1 added to submatrix */ FLA_BOTTOM );

    // Free local objects...
}
```

Figure 4: Lazy block-row oriented triangular Sylvester equation solver implemented using FLAME.

The algorithm is stated in Figure 2 and is equivalent to the traditional eager variant shown in Figure 3 (right).

3.1.3 Proving correctness and cost

The following theorems proves the correctness of the the lazy and eager block-row oriented algorithms and present their computational cost.

Theorem 1 *The lazy and eager block-row oriented triangular Sylvester equation solvers in Figure 2 overwrite matrix C with the solution of the triangular Sylvester equation $AX + XB = C$.*

Proof. *The following table summarizes the contents of matrix C at various stages of the lazy algorithm:*

Stage	*Contents*	*Comment*
Before entering the loop	$\left(\frac{C_T}{X_B}\right)$	*where X_B has 0 rows, and thus $C_T = C$*
At the beginning of iteration k, $k = 0, 1, 2, \ldots, m/b_m - 1$	$\left(\frac{C_T}{X_B}\right)$	*where X_B has kb_m rows and C_T has $m - kb_m$ rows.*
Upon exiting the loop	$\left(\frac{C_T}{X_B}\right)$	*where C_T has 0 rows and thus $X_B = X$*

Notice also that the algorithm advances by $b_m > 0$ rows at each iteration, until C_T is $0 \times n$, and thus it is guaranteed to terminate. At the end of the final iteration the loop-invariant will also hold and therefore the contents of C will be those of $\Omega(X_B) = X$.

The proof for the eager algorithm is similar. □

Theorem 2 *The lazy and eager block-row oriented triangular Sylvester equation solvers in Figure 2 both require $m^2n + mn^2$ floating point operations.*

Proof. *We prove the theorem for the case where b_m is constant.*

In the algorithms in Figure 2 the size of C_B increases from $b_m \times n$ to $(m - b_m) \times n$, while the size of A_{BR} increases from $b_m \times b_m$ to $(m - b_m) \times (m - b_m)$. Assuming C_B currently is $k_m \times n$, and A_{BR} is thus currently

$k_m \times k_m$, the different parts of the matrices have the following dimensions:

$$
\begin{array}{c|c||c}
\overbrace{\phantom{A_{00}}}^{\bar{m}} & \overbrace{\phantom{A_{01}}}^{b_m} & \overbrace{\phantom{A_{02}}}^{k_m} \\
A_{00} & A_{01} & A_{02} \\ \hline
0 & A_{11} & A_{12} \\ \hline\hline
0 & 0 & A_{22}
\end{array}
\quad
\begin{array}{l}
\\
\}\bar{m} \\
\}b_m \\
\}k_m
\end{array}
\qquad
\begin{array}{c}
\overbrace{}^{n} \\
C_0 \\ \hline
C_1 \\ \hline\hline
C_2
\end{array}
\quad
\begin{array}{l}
\\
\}\bar{m} \\
\}b_m \\
\}k_m
\end{array}
$$

Here, $\bar{m} = m - k_m - b_m$.

The number of floating point operations required to move the computation forward by b_m rows in the lazy and eager versions of the algorithm is given by

$$
\begin{array}{rll}
\bar{C}_1 \equiv & C_1 \leftarrow C_1 - A_{12}X_2 & 2b_m k_m n \\
\Omega(X_1) \equiv & A_{11}X_1 + X_1 B = C_1 & b_m^2 n + b_m n^2 \\
\bar{C}_0 \equiv & C_0 \leftarrow C_0 - A_{01}X_1 & 2b_m k_m \bar{n}
\end{array}
$$

For simplicity we neglect the lower order terms in the computation of the cost of the algorithms. If we consider the algorithm to iterate for $k = 0, 1, 2, \ldots, m/b_m - 1$, then $k_m = kb_m$. Table 1 reports the cost of these three operations and the overall cost of the algorithms, proving the theorem.

Operation	Cost	
	Lazy variant	Eager variant
$\bar{C}_1$	$\sum_{k=0}^{m/b_m-1} (2b_m k_m n) \approx m^2 n$	–
$\Omega(X_1)$	$\sum_{k=0}^{m/b_m-1} (b_m^2 n + b_m n^2) \approx mn^2$	$\sum_{k=0}^{m/b_m-1} (b_m^2 n + b_m n^2) \approx mn^2$
$\bar{C}_0$	–	$\sum_{k=0}^{m/b_m-1} (2b_m k_m \bar{n}) \approx m^2 n$
Total	$m^2 n + mn^2$	$m^2 n + mn^2$

Table 1: Cost of the lazy and eager block-row oriented triangular Sylvester equation solvers derived from R1 (lazy) and R2 (eager).

$\square$

Notice that if the triangular Sylvester equations arising in the block-row oriented algorithms are solved using a traditional, non-blocked solver, $m \approx n$, and $b_m \ll m$, half the computation is in operations involving smaller Sylvester equations and the other half is in matrix multiplications.

3.2 Block-column oriented solvers

By partitioning B instead of A, we obtain "symmetric" algorithms which compute the solution X by column blocks.

4 A Family of Blocked Algorithms

In this section we show that a partitioning of both coefficient matrices leads to a family of sixteen different blocked solvers.

Consider starting our derivation of blocked algorithms by partitioning both coefficient matrices into four quadrants

$$A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array} \right), \quad B \rightarrow \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array} \right),$$

where A_{BR} is a $k_m \times k_m$ block and B_{TL} is a $k_n \times k_n$ block. Accordingly, we next apply a conformal partition to X and C

$$X \rightarrow \left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array} \right), \quad C \rightarrow \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array} \right),$$

where X_{BL} and C_{BL} are $k_m \times k_n$ blocks.

Now, (1) becomes

$$\begin{array}{l} \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array} \right) \left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array} \right) + \left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array} \right) \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array} \right) \\ \quad = \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array} \right), \end{array}$$

where, multiplying out the left-hand-side of the matrix equation, we obtain the following equalities

$$\begin{array}{rcl} A_{TL}X_{TL} + X_{TL}B_{TL} & = & C_{TL} - A_{TR}X_{BL}, \\ A_{TL}X_{TR} + X_{TR}B_{BR} & = & C_{TR} - A_{TR}X_{BR} - X_{TL}B_{TR}, \\ A_{BR}X_{BL} + X_{BL}B_{TL} & = & C_{BL}, \\ A_{BR}X_{BR} + X_{BR}B_{BR} & = & C_{BR} - X_{BL}B_{TR}. \end{array}$$

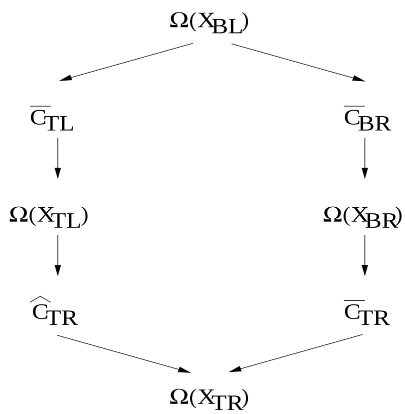

Figure 5: Data dependencies for the partitioned triangular Sylvester matrix equation.

We will designate these individual operations as follows:

$$\begin{array}{rcl} \Omega(X_{TL}) & \equiv & A_{TL}X_{TL} + X_{TL}B_{TL} = C_{TL}, \\ \Omega(X_{TR}) & \equiv & A_{TL}X_{TR} + X_{TR}B_{BR} = C_{TR}, \\ \Omega(X_{BL}) & \equiv & A_{BR}X_{BL} + X_{BL}B_{TL} = C_{BL}, \\ \Omega(X_{BR}) & \equiv & A_{BR}X_{BR} + X_{BR}B_{BR} = C_{BR}, \\ \bar{C}_{TL} & \equiv & C_{TL} \leftarrow C_{TL} - A_{TR}X_{BL}, \\ \bar{C}_{TR} & \equiv & C_{TR} \leftarrow C_{TR} - A_{TR}X_{BR}, \\ \hat{C}_{TR} & \equiv & C_{TR} \leftarrow C_{TR} - X_{TL}B_{TR}, \\ \bar{C}_{BR} & \equiv & C_{BR} \leftarrow C_{BR} - X_{BL}B_{TR}. \end{array} \tag{3}$$

The dependencies among the operations induce a certain order: the first operation that must be performed is solving $\Omega(X_{BL})$; after that, only the updates $\bar{C}_{TL}$ or $\bar{C}_{BR}$ are possible, and so on. Figure 5 shows graphically these data dependencies.

In order to derive blocked solvers for the triangular Sylvester equation, we could start by considering all possible cases where some of the operations in (3) have already been performed. However, with eight different operations, we have $\sum_{i=0}^{8} \left(\begin{array}{c} 8 \\ i \end{array} \right)$ possibilities! Luckily, due to the dependencies, there are only sixteen valid cases, as summarized in Table 2. Notice that we label six of these cases as symmetric.

Case	Operations performed (Current contents of C)	Operations performed (Current contents of C)	Symm. Case
C1	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & C_{BR} \end{array}\right)$		
C2	$\left(\begin{array}{c\|c} \bar{C}_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & C_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & \bar{C}_{BR} \end{array}\right)$	C11
C3	$\left(\begin{array}{c\|c} \bar{C}_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & \bar{C}_{BR} \end{array}\right)$		
C4	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & C_{TR} \\ \hline \Omega(X_{BL}) & C_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	C12
C5	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & \hat{C}_{TR} \\ \hline \Omega(X_{BL}) & C_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} C_{TL} & \bar{C}_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	C13
C6	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & C_{TR} \\ \hline \Omega(X_{BL}) & \bar{C}_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} \bar{C}_{TL} & C_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	C14
C7	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & C_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$		
C8	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & \hat{C}_{TR} \\ \hline \Omega(X_{BL}) & \bar{C}_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} \bar{C}_{TL} & \bar{C}_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	C15
C9	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & \bar{C}_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & \hat{C}_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$	C16
C10	$\left(\begin{array}{c\|c} \Omega(X_{TL}) & \hat{C}_{TR}, \bar{C}_{TR} \\ \hline \Omega(X_{BL}) & \Omega(X_{BR}) \end{array}\right)$		

Table 2: Valid intermediate cases.

Repartition the matrices of the equation into nine blocks, as follows:

$$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c\|c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} B_{00} & B_{01} & B_{02} \\ \hline\hline 0 & B_{11} & B_{12} \\ \hline 0 & 0 & B_{22} \end{array}\right),$$

where A_{11} and B_{11} are, respectively, $b_m \times b_m$ and $b_n \times b_n$ blocks. Conformally, repartition

$$
\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline\hline X_{20} & X_{21} & X_{22} \end{array}\right),
$$

$$
\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline\hline C_{20} & C_{21} & C_{22} \end{array}\right),
$$

where X_{11} and C_{11} are both $b_m \times b_n$ blocks. Notice that, again, the double lines mark how far (the boundaries of) the computation has progressed. For the blocked algorithms, the solvers march in matrices X and C from the bottom-left corner to the top-right one, moving b_m rows and b_n columns at each iteration. In A, the solvers towards the top-left corner, while in B the direction is towards the bottom-right corner. Lower triangular matrices A and/or B would produce all other possibilities in the direction of the computation.

We next illustrate the derivation of a blocked algorithm resulting from a specific case, here, C2. In this case, the current contents of C are given by

$$
\left(\begin{array}{c||c} C_{TL} - A_{TR}X_{BL} & C_{TR} \\ \hline\hline X_{BL} & C_{BR} \end{array}\right) = \left(\begin{array}{c||c} \left(\begin{array}{c} C_{00} - A_{02}X_{20} \\ \hline C_{10} - A_{12}X_{20} \end{array}\right) & \left(\begin{array}{c|c} C_{01} & C_{02} \\ \hline C_{11} & C_{12} \end{array}\right) \\ \hline\hline X_{20} & \left(\begin{array}{c|c} C_{21} & C_{22} \end{array}\right) \end{array}\right).
$$

Consider now the following repartitioning which corresponds to the next stage where the computation has moved forward by a block of dimension $b_m \times b_n$

$$
\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right),
$$

$$
\left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} B_{00} & B_{01} & B_{02} \\ \hline 0 & B_{11} & B_{12} \\ \hline\hline 0 & 0 & B_{22} \end{array}\right),
$$

$$
\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} X_{00} & X_{01} & X_{02} \\ \hline\hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array}\right),
$$

$$
\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} C_{00} & C_{01} & C_{02} \\ \hline\hline C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array}\right).
$$

With this new repartitioning, we wish the contents of C to become

$$\left(\begin{array}{c||c} C_{TL} - A_{TR}X_{BL} & C_{TR} \\ \hline\hline X_{BL} & C_{BR} \end{array}\right) = \left(\begin{array}{c||c} \left(\begin{array}{cc|cc} C_{00} & -(A_{02}X_{20} & C_{01} & -(A_{02}X_{21} \\ & +A_{01}X_{10}) & & +A_{01}X_{11}) \end{array}\right) & C_{02} \\ \hline\hline \left(\begin{array}{c|c} X_{10} & X_{11} \\ \hline X_{20} & X_{21} \end{array}\right) & \left(\begin{array}{c} C_{12} \\ \hline C_{22} \end{array}\right) \end{array}\right).$$

Therefore, in order to move the computation forward in case C2 we need to perform the operations shown in Figure 6 (center).

Theorem 3 *The algorithms in Figure 6 overwrite matrix C with the solution of the triangular Sylvester equation $AX + XB = C$.*

Proof. The proof of this theorem is much like the one given for Theorem 1. □

The next theorem derives the cost of the blocked triangular Sylvester equation solver presented in the figure.

Theorem 4 *Given that matrices X and C are $m \times n$, A is an $m \times m$ triangular matrix and B is an $n \times n$ triangular matrix, the blocked triangular Sylvester equation solver in Figure 6 requires $m^2n + mn^2$ floating point operations.*

Proof. We prove the theorem for the case where the blocks sizes b_m and b_n are constant.

In the algorithm presented in Figure 6 the size of C_{BL} increases from $b_m \times b_n$ to $(m - b_m) \times (n - b_n)$, while the size of A_{BR} increases from $b_m \times b_m$ to $(m - b_m) \times (m - b_m)$, and that of B_{TL} increases from $b_n \times b_n$ to $(n - b_n) \times (n - b_n)$. Assuming C_{BL} is currently $k_m \times k_n$, and A_{BR}, B_{TL} are currently $k_m \times k_m$ and $k_n \times k_n$, respectively, the different parts of the matrices have the following dimensions:

$$\begin{array}{c|c||c} \overbrace{A_{00}}^{\bar{m}} & \overbrace{A_{01}}^{b_m} & \overbrace{A_{02}}^{k_m} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array} \begin{array}{l} \}\bar{m} \\ \}b_m \\ \}k_m \end{array} \qquad \begin{array}{c||c|c} \overbrace{B_{00}}^{k_n} & \overbrace{B_{01}}^{b_n} & \overbrace{B_{02}}^{\bar{n}} \\ \hline\hline 0 & B_{11} & B_{12} \\ \hline 0 & 0 & B_{22} \end{array} \begin{array}{l} \}k_n \\ \}b_n \\ \}\bar{n} \end{array}$$

$$\begin{array}{c||c|c} \overbrace{C_{00}}^{k_n} & \overbrace{C_{01}}^{b_n} & \overbrace{C_{02}}^{\bar{n}} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline\hline C_{20} & C_{21} & C_{22} \end{array} \begin{array}{l} \}\bar{m} \\ \}b_m \\ \}k_m \end{array}$$

Algorithm 2 $C \leftarrow X$, *where* $AX + XB = C$
(Blocked)

partition

$$A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), \quad B \rightarrow \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right), \quad C \rightarrow \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right),$$

where A_{BR}, B_{TL}, **and** C_{BL} **are** 0×0

do until C_{TR} **is** 0×0

determine block sizes b_m *and* b_n

repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} B_{00} & B_{01} & B_{02} \\ \hline\hline 0 & B_{11} & B_{12} \\ \hline 0 & 0 & B_{22} \end{array}\right),$$

$$\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline\hline C_{20} & C_{21} & C_{22} \end{array}\right),$$

where A_{11} **is** $b_m \times b_m$, B_{11} **is** $b_n \times b_n$, **and** C_{11} **is** $b_m \times b_n$

C1 variant:	*C2 variant:*	*C3 variant:*
$C_{10} \leftarrow C_{10} - A_{12}X_{20}$	$C_{10} \leftarrow X_{10}$, *where*	$C_{10} \leftarrow X_{10}$, *where*
$C_{10} \leftarrow X_{10}$, *where*	$A_{11}X_{10} + X_{10}B_{00} = C_{10}$	$A_{11}X_{10} + X_{10}B_{00} = C_{10}$
$A_{11}X_{10} + X_{10}B_{00} = C_{10}$	$C_{11} \leftarrow C_{11} - X_{10}B_{01}$	$C_{11} \leftarrow C_{11} - X_{10}B_{01}$
$C_{11} \leftarrow C_{11} - X_{10}B_{01}$	$C_{21} \leftarrow C_{21} - X_{20}B_{01}$	$C_{21} \leftarrow X_{21}$, *where*
$C_{21} \leftarrow C_{21} - X_{20}B_{01}$	$C_{21} \leftarrow X_{21}$, *where*	$A_{22}X_{21} + X_{21}B_{11} = C_{21}$
$C_{21} \leftarrow X_{21}$, *where*	$A_{22}X_{21} + X_{21}B_{11} = C_{21}$	$C_{11} \leftarrow C_{11} - A_{12}X_{21}$
$A_{22}X_{21} + X_{21}B_{11} = C_{21}$	$C_{11} \leftarrow C_{11} - A_{12}X_{21}$	$C_{11} \leftarrow X_{11}$, *where*
$C_{11} \leftarrow C_{11} - A_{12}X_{21}$	$C_{11} \leftarrow X_{11}$, *where*	$A_{11}X_{11} + X_{11}B_{11} = C_{11}$
$C_{11} \leftarrow X_{11}$, *where*	$A_{11}X_{11} + X_{11}B_{11} = C_{11}$	$C_{00} \leftarrow C_{00} - A_{01}X_{10}$
$A_{11}X_{11} + X_{11}B_{11} = C_{11}$	$C_{00} \leftarrow C_{00} - A_{01}X_{10}$	$C_{01} \leftarrow C_{01} - A_{01}X_{11}$
	$C_{01} \leftarrow C_{01} - A_{01}X_{11}$	$C_{01} \leftarrow C_{01} - A_{02}X_{21}$
	$C_{01} \leftarrow C_{01} - A_{02}X_{21}$	$C_{12} \leftarrow C_{12} - X_{10}B_{02}$
		$C_{12} \leftarrow C_{12} - X_{11}B_{12}$
		$C_{22} \leftarrow C_{22} - X_{21}B_{12}$

continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} B_{00} & B_{01} & B_{02} \\ \hline 0 & B_{11} & B_{12} \\ \hline\hline 0 & 0 & B_{22} \end{array}\right),$$

$$\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} C_{00} & C_{01} & C_{02} \\ \hline\hline C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array}\right)$$

enddo

Figure 6: Blocked triangular Sylvester equation solvers derived from C1, C2, and C3.

Here, $\bar{m} = m - k_m - b_m$*,* $\bar{n} = n - k_n - b_n$*.*

The number of floating point operations required to move the computation forward is given by

$$\begin{array}{rll}
\Omega(X_{10}) \equiv & A_{11}X_{10} + X_{10}B_{00} = C_{10} & b_m^2 k_n + b_m k_n^2 \\
& C_{11} \leftarrow C_{11} - X_{10}B_{01} & 2b_m b_n k_n \\
\bar{C}_{21} \equiv & C_{21} \leftarrow C_{21} - X_{20}B_{01} & 2b_n k_m k_n \\
\Omega(X_{21}) \equiv & A_{22}X_{21} + X_{21}B_{11} = C_{21} & b_n^2 k_m + b_n k_m^2 \\
& C_{11} \leftarrow C_{11} - A_{12}X_{21} & 2b_m b_n k_m \\
& A_{11}X_{11} + X_{11}B_{11} = C_{11} & b_m^2 b_n + b_m b_n^2 \\
\bar{C}_{00} \equiv & C_{00} \leftarrow C_{00} - A_{01}X_{10} & 2b_m k_n \bar{m} \\
& C_{01} \leftarrow C_{01} - A_{01}X_{11} & 2b_m b_n \bar{m} \\
\bar{C}_{01} \equiv & C_{01} \leftarrow C_{01} - A_{02}X_{21} & 2b_n k_m \bar{m}
\end{array}$$

For simplicity we neglect the lower order terms in the computation of the cost of the algorithm, which leads us to consider only the operations denoted as $\Omega(X_{10})$*,* $\bar{C}_{21}$*,* $\Omega(X_{21})$*,* $\bar{C}_{00}$*, and* $\bar{C}_{01}$*. If we consider the algorithm to iterate for* $k = 0, 1, 2, \ldots, \max(m/b_m, n/b_n) - 1$*, then* $k_m = kb_m$ *and* $k_n = kb_n$*. Table 3 reports the cost of these five operations and the overall cost of the algorithm, proving the theorem.*

□

As could be expected, the blocked algorithm presents the same computational cost as the serial solver. The blocked algorithms for the remaining 15 cases are obtained by simply deriving the set of operations that will satisfy the loop-invariant in each case, and they all can be shown to present the same cost.

5 Heuristics

In the previous section, we derived a large number of algorithms for the solution of the triangular Sylvester equation. The question now becomes how to design a near-optimal implementation. In this section, we present both theoretical and practical insights that help guide the way.

First, let us review observations regarding blocked algorithms in general. All are designed to spend a substantial part of the computation in the matrix multiplication (GEMM) kernel. Thus, it makes sense to pick block sizes b_m and b_n that allow the individual calls to the matrix-matrix multiply to attain the highest performance, subject to other constraints.

Consider the matrix multiplication $C \leftarrow AB + C$, where A is $m \times k$, B is $k \times n$, and then C is $m \times n$. In [9], it is shown that, for architectures

Operation	Cost	
	$m/b_m \geq n/b_n$	$n/b_n > m/b_m$
$\bar{C}_{21}$	$\sum_{k=0}^{n/b_n-1}(2b_n k_m k_n) \approx \frac{2b_m n^3}{3b_n}$	$\sum_{k=0}^{m/b_m-1}(2b_n k_m k_n) + \sum_{k=m/b_m}^{n/b_n-1}(2b_n k_n m) \approx mn^2 - \frac{b_n^2 m^3}{3b_m^2}$
$\bar{C}_{00}$	$\sum_{k=0}^{n/b_n-1}(2b_m k_n \bar{m}) + \sum_{k=n/b_n}^{m/b_m-1}(2b_m \bar{m} n) \approx mn^2 - \frac{b_m mn^2}{b_n} + \frac{b_m^2 n^3}{3b_n^2}$	$\sum_{k=0}^{m/b_m-1}(2b_m k_n \bar{m}) \approx \frac{b_n m^3}{3b_m}$
$\bar{C}_{01}$	$\sum_{k=0}^{n/b_n-1}(2b_n k_m \bar{m}) \approx \frac{b_m mn^2}{b_n} - \frac{2b_m^2 n^3}{3b_n^2}$	$\sum_{k=0}^{m/b_m-1}(2b_m k_n \bar{m}) \approx \frac{b_n m^3}{3b_m}$
$\Omega(X_{10})$	$\sum_{k=0}^{n/b_n-1}(b_m k_n^2) + \sum_{k=n/b_n}^{m/b_m-1}(b_m n^2) \approx mn^2 - \frac{2b_m n^3}{3b_n}$	$\sum_{k=0}^{m/b_m-1}(b_m k_n^2) \approx \frac{b_n^2 m^3}{3b_m^2}$
$\Omega(X_{21})$	$\sum_{k=0}^{n/b_n-1}(b_n k_m^2) \approx \frac{b_m^2 n^3}{3b_n^2}$	$\sum_{k=0}^{m/b_m-1}(b_n k_m^2) + \sum_{k=m/b_m}^{n/b_n-1}(b_n m^2) \approx m^2 n - \frac{2b_n m^3}{3b_m}$
Total	$m^2 n + mn^2$	$m^2 n + mn^2$

Table 3: Cost of the blocked triangular Sylvester equation solver derived from C2.

with two levels of cache memory, there are two block sizes that influence the performance of GEMM: b_1 and b_2, which are related to the size of the L1 and L2 caches, respectively. In Table 4 we show how the three matrix dimensions, m, n, and k affect performance of GEMM. Where it says "large" in the table, the larger the dimensions, the better the performance.

Logic suggests that we attempt to minimize the amount of computation in the solution of the smaller Sylvester equations that show up in the body of the loop, thereby maximizing the amount of computation in GEMM at this level (notice that each of these smaller Sylvester equations can be solved using the same algorithm, generating a recursion with multiple levels). In Section 3 we mentioned that the block-row oriented algorithms spend approximately half of the computation in these sub-problems. By symmetry,

Shape	Performance
2 of m, n, k large 1 of $m, n, k = b_2$	Best
1 of m, n, k large 2 of $m, n, k = b_2$	$\uparrow$
2 of $m, n, k = b_2$ 1 of $m, n, k = b_1$	$\downarrow$
1 of $m, n, k = b_2$ 2 of $m, n, k = b_1$	Good
$1 < m, n, k \leq b_1$	Worse
1 or more of $m, n, k = 1$	Worst

Table 4: Factors affecting the performance of the matrix multiplication.

the same is true for the block-column oriented algorithms. We now show that, if the block sizes are chosen carefully, the blocked algorithms spend only a third of the computation in the sub-problems.

Consider the part of the total computational cost that is spent in solving the Sylvester equations in the body of the loop if we apply, e.g., the algorithm derived for C2, with $b_m, b_n \ll m, n$. (Here we can assume without loss of generality that $m/b_m \geq n/b_n$.) At each iteration of the algorithm we need to solve two large Sylvester equations, for $\Omega(X_{10})$ and $\Omega(X_{21})$, with an overall computational cost (see Table 3) of $mn^2 + \frac{b_m n^3}{3b_n}\left(\frac{bm}{bn} - 2\right)$ flops (floating-point arithmetic operations). The question therefore becomes what are the values of b_m and b_n that minimize this value, i.e., $\min_{\{b_m, b_n\}} \frac{b_m}{b_n}\left(\frac{bm}{bn} - 2\right)$. This minimum is attained for $b_m = b_n$ and, in this case, approximately a third of the computation is spent in the sub-problems. We can conclude that we should pursue the use of square block sizes.

Let us revisit the observations made regarding the matrix-matrix multiply. The following table explains a simple heuristic:

Heuristic 1			
Case		Strategy	Comment
m	n		
large	large	blocked with $b_m = b_n = b_2$	Calls to GEMM involve two large dimensions and one equal to b_2.
b_2	large	block-column with $b_n = b_2$	Calls to GEMM involve one large dimension and two equal to b_2.
large	b_2	block-row with $b_m = b_2$	Calls to GEMM involve one large dimension and two equal to b_2.
b_2	b_2	blocked with $b_m = b_n = b_1$	Calls to GEMM involve two dimensions equal to b_2 and one equal to b_1.
b_1	b_2	block-column with $b_n = b_1$	Calls to GEMM involve one dimension equal to b_2 and two equal to b_1.
b_1	b_1	any	Note that little computation is left; blocking is less important.

By choosing different blocked and/or block-row and block-column oriented algorithms at the different levels, a large family of different hybrid algorithms is specified by the above table.

We now present a second heuristic. Consider the top-level blocking. If one takes $b_m = b_n = b_2$, then two thirds of the computation will be in matrix multiplication. If, on the other hand, one considers a block-row oriented algorithm, with block size $b_m = b_2$, then only half of the computation is in terms of the matrix-matrix multiply. However, in this second alternative, one of the dimensions involved in the matrix-matrix multiply always equals b_2, one always equals n, and the third ranges from small to large (C2 acts as an eager algorithm). A similar observation can be made for a block-column oriented algorithm. By contrast, if a blocked algorithm is used, one dimension always equals b_2 while two of the dimensions range from small to large or vice-versa. Furthermore, when blocked algorithms are used, a larger number of calls to GEMM are made. Thus, *in practice*, we can expect the calls to GEMM employed by the block-row or block-column algorithms to attain higher performance than the calls employed by the blocked algorithms when $b_m = b_n = b_2$. Thus, a second heuristic becomes

Heuristic 2			
Case		Strategy	Comment
m	n		
large	large	block-row with $b = b_2$	Calls to GEMM involve two large dimensions and one equal to b_2.
b_2	large	block-column with $b = b_2$	Calls to GEMM involve one large dimension and two equal to b_2.
large	b_2	block-row with $b = b_2$	Calls to GEMM involve one large dimension and two equal to b_2.
b_2	b_2	block-row with $b = b_1$	Calls to GEMM involve two dimensions equal to b_2 and one equal to b_1.
b_1	b_2	block-column with $b = b_1$	Calls to GEMM involve one dimension equal to b_2 and two equal to b_1.
b_1	b_1	any	Note that little computation is left; blocking is less important.

This second heuristic does not require blocked algorithms. However, we now show that by picking b_m and b_n carefully, some blocked algorithms can be used to implement the second heuristic in a particularly elegant fashion. Notice that by setting $b_m = m$ or $b_n = n$, some of the cases of the blocked algorithms become either block-column or block-row algorithms, respectively:

Resulting algorithm	Cases	
	$b_m = m$	$b_n = n$
Lazy block-column	**C1**, **C2**, C4, C5	–
Eager block-column	**C3**, C6, C8, **C11**	–
Lazy block-row	–	**C1**, **C11**, C12, C13
Eager block-row	–	**C2**, **C3**, C14, C15

Notice that four variants, C1, C2, C3, and C11, have the property that by picking the block sizes carefully, they can become either block-row or block-column algorithms. For these variants, the following strategy will automatically generate an algorithm which conforms the second heuristic: recursively call the given algorithm with, progressively, the block sizes $b_2 \times n$, $b_2 \times b_2$, $b_1 \times b_2$, $b_1 \times b_1$, followed by some strategy for solving the small $b_1 \times b_1$ Sylvester equation sub-problems that remain at the lowest level of the recursion. For example, we can employ the same algorithm to reduce the sub-problem to a certain size, $b_0 \times n$, apply an additional level of recursion to reduce it further to $b_0 \times b_0$, and solve this square sub-problems using a non-blocked algorithm.

6 Experimental Results

In this section we report the performance attained by our algorithms as the rate of computations achieved in millions of flops per second (MFLOPS/sec). We consider here triangular Sylvester equations of dimension $m \times n$, and an operation count of $m^2n + mn^2$ flops.

We report performance on an Intel (R) Pentium III (650 MHz) processor with a 16 Kbyte L1 data cache and a 256 Kbyte L2 cache running RedHat Linux 7.1. All computations were performed in 64-bit (double precision) arithmetic, and the same options were used when compiling the different implementations.

We analyze performance for five different implementations, indicated by the curves marked as follows:

`unb`: Eager column-oriented implementation using Fortran-77. (We also implemented eager/lazy and row-/column-oriented variants, but the results were inferior for those). Due to the poor performance of this approach we only report results for the smaller problem sizes.

`Trad. blocked`: Row-eager/column-eager blocked implementation of the solver using using Fortran-77, similar to that in Figure 1. (We also implemented all other variants; their performance proved inferior).

`C1, C2, and C3`: Our implementations of the solvers using FLAME and the heuristics described in the previous section. At the lowest level of the recursion, the sub-problems are computed using the unblocked (`unb`) solver. For ITXGEMM we determined b_0, b_1, and b_2 experimentally to be 16, 64, and 128, respectively. For ATLAS these values were, respectively, 20, 40, and 120. The reason for the difference between the block sizes for ITXGEMM and ATLAS lies in the details of the implementations of these packages [9].

Although we implemented all blocked, block-row, and block-column algorithms, we only present results for C1, C2, and C3, as C11 is symmetric with C2, the remaining blocked algorithms obtained slightly worse performance. Also we already argued that the block-row and block-column algorithms are special cases of C1, C2, C3, and C11.

Figure 7 reports the performance for these algorithms using ITXGEMM (top) and ATLAS R3.2 (bottom). We also include the performance of the GEMM routine for a matrix multiplication with $k = b_2$ (a rank-k update) for reference. We only report the results using the second heuristic described in the previous section, with block sizes $b_2 \times n$, $b_2 \times b_2$, $b_1 \times b_2$, $b_1 \times b_1$, $b_0 \times b_1$, and $b_0 \times b_0$, for the different levels of the recursion.

For the largest problem sizes, the performance attained by our algorithms using the second heuristic is close to 90% of that achieved for the matrix-matrix multiply kernel in ITXGEMM (around 530 MFLOPS/sec.). The same kernel from ATLAS achieves a somewhat lower performance (510 MFLOPS/sec.), which partially explains the lower performance of the solvers when this GEMM kernel is employed.

Figure 8 reports the performance of the best solver (that derived from C3) using both Heuristic 1 and Heuristic 2. It appears that on this architecture the algorithms benefit from the larger matrix sizes involved in the calls to GEMM in Heuristic 2.

Although not reported here, our solvers, when linked to ATLAS R3.2, obtain performance similar to that reported in [12] when we adjust for the different clock rate of the processor on which they performed their experiments.

7 Concluding Remarks

In this paper, we have made a number of contributions to the solution of the triangular Sylvester equation. These include:

- The systematic derivation and proof of correctness of a family of algorithms using the FLAME approach.
- The implementation of the family using the FLAME library.
- A heuristic for composing high-performance implementations from members of the family of algorithms.
- A demonstration of excellent performance.
- Altogether, we have presented dozens of new algorithms for the solution of this problem.

Many of our observations can be extended to blocked algorithms for dense linear algebra operations in general. These include:

- The FLAME approach is a powerful tool for the derivation of provably correct blocked algorithms.
- The FLAME library provides a prototype environment for the rapid implementation of such algorithms.

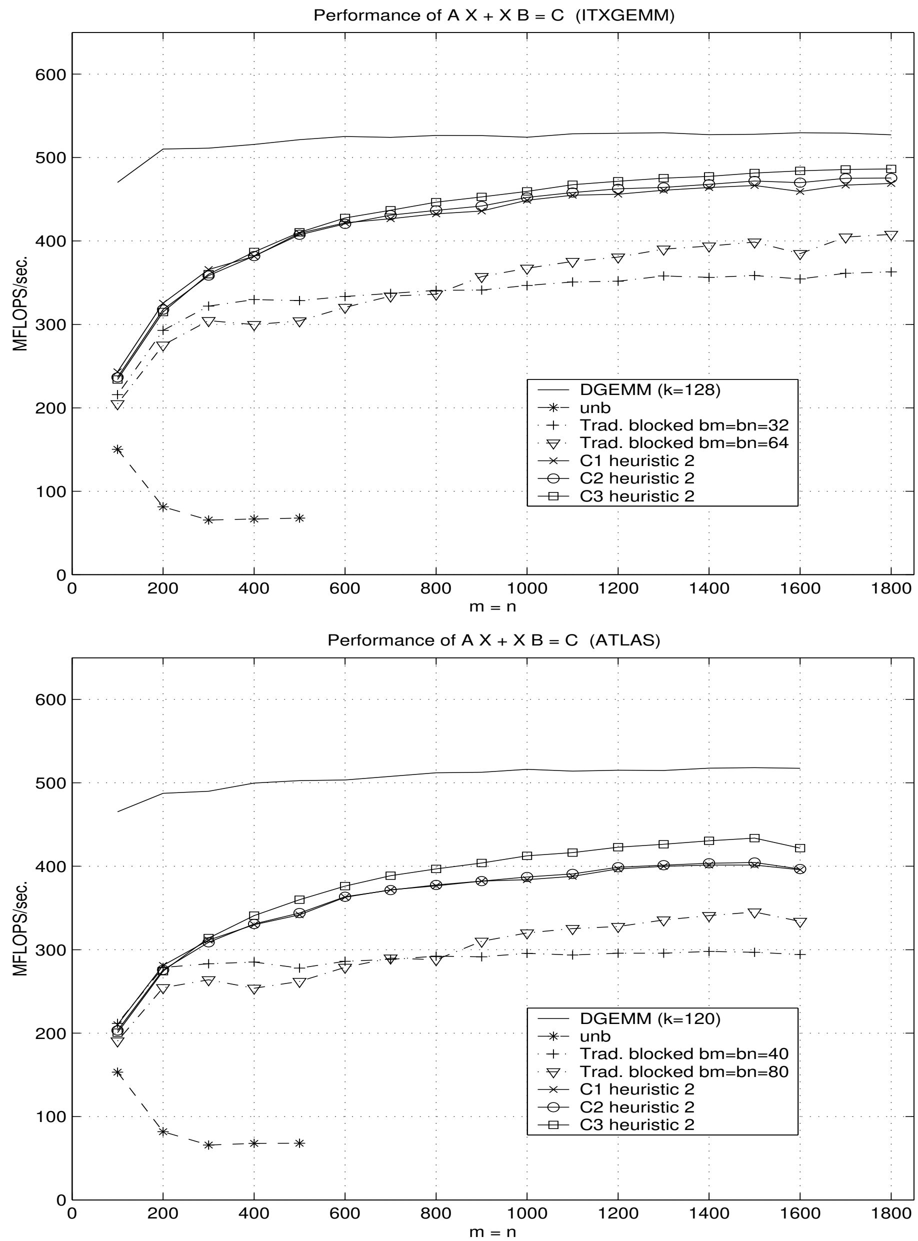

Figure 7: Performance of the triangular Sylvester equation solvers using ITXGEMM (top) and ATLAS R3.2 (bottom).

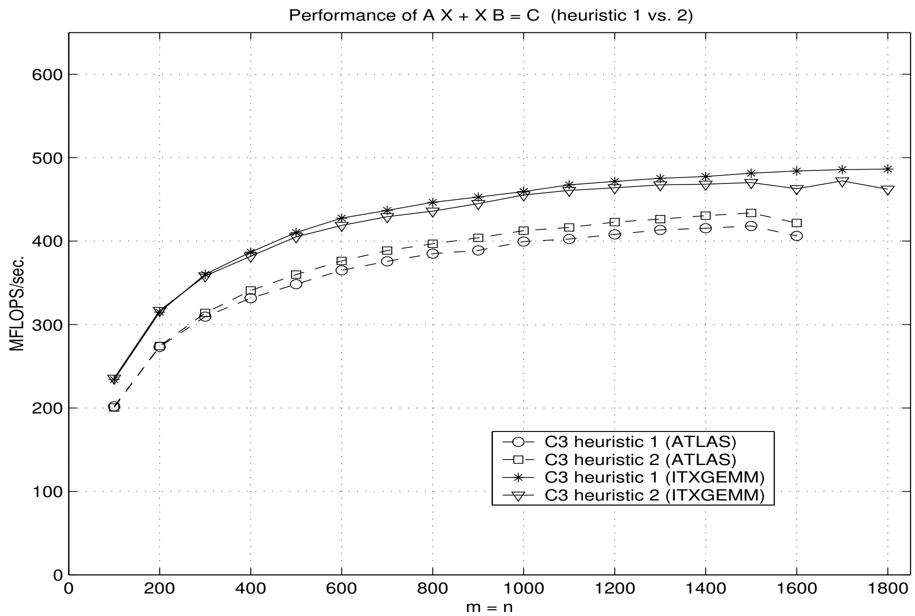

Figure 8: Performance of the triangular Sylvester equation solvers using heuristic 1 and heuristic 2.

- The heuristics developed for composing members of the family are likely to be extensible to blocked algorithms in general.
- The FLAME approach and library naturally support the hybrid recursive/iterative algorithms that are both general and required for near-optimal implementation.

Acknowledgments

Primary support for this work came from the Visiting Researcher program of the Texas Institute for Computational and Applied Mathematics (TICAM).

We thank John Gunnels for his comments on a preliminary version of this paper.

Additional Information

For additional information on FLAME please visit

http://www.cs.utexas.edu/users/flame/

References

[1] F.A. Aliev and V.B. Larin. *Optimization of Linear Control Systems: Analytical Methods and Computational Algorithms*, volume 8 of *Stability and Control: Theory, Methods and Applications.* Gordon and Breach, 1998.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, second edition, 1995.

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

[4] J. Bilmes, K. Asanovic, C.W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performace, ANSI C coding methodology. In *Proc. of the International Conference on SuperComputing, ACM SIGARC*, 1997.

[5] D. Calvetti and L. Reichel. Application of ADI iterative methods to the restoration of noisy images. *SIAM J. Matrix Anal. Appl.*, 17:165–186, 1996.

[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.*, 16:1–17, 1990.

[7] G. H. Golub, S. Nash, and C. F. Van Loan. A Hessenberg–Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control*, AC-24:909–913, 1979.

[8] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[9] J. Gunnels, G. Henry, and R.A. van de Geijn. A family of high-performance matrix multiplication algorithms. In V.N. Alexander,

J. Dongarra, B.A. Julianno, R.S. Renner, and C.J. Kenneth Tan, editors, *Computational Science - ICCS 2001, Part I, Lecture Notes in Computer Science 2073*, pages 51–60, 2001.

[10] J. Gunnels, G. Henry, and R.A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Trans. on Math. Software*, 2001. To appear.

[11] S.J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[12] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular matrix equations - part I: One-sided and coupled Sylvester equations. Department of Computing Science and HPC2N Report UMINF-01.05, Umeå University, 2001.

[13] B. Kågström and P. Poromaa. Distributed and shared memory block algorithms for the triangular Sylvester equation with sep^{-1} estimator. *SIAM J. Matrix Anal. Appl.*, 13(1):90–101, 1992.

[14] V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, NY, 1996.

[15] R. C. Whaley and J. Dongarrra. Automatically tuned linear algebra software. In *Proc. of SC98*, 1998.