# Formal Derivation of Algorithms: The Triangular Sylvester Equation

ENRIQUE S. QUINTANA-ORTÍ
Universidad Jaume I
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

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 (R) III processor that is competitive with that of recursive algorithms reported previously in the literature for this operation.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Algorithm Design and Analysis; Efficiency; User Interfaces*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain Specific Achitectures*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Software Libraries*

General Terms: Algorithms; Design; Theory; Performance

Additional Key Words and Phrases: Formal derivation; libraries, linear algebra; control theory; Sylvester equations

## 1. INTRODUCTION

In a recent paper the Formal Linear Algebra Methods Environment (FLAME) was introduced [Gunnels et al. 2001]. 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

Authors' addresses: Enrique S. Quintana-Ortí, Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, `quintana@icc.uji.es`. Robert A. van de Geijn, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, `rvdg@cs.utexas.edu`.

to illustrate the basic techniques.

In this paper, we demonstrate the versatility of FLAME by concentrating on a more complex linear algebra operation, the solution of 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., [Bartels and Stewart 1972; Golub et al. 1979; Hammarling 1982].

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., [Aliev and Larin 1998; Calvetti and Reichel 1996; Golub and Van Loan 1996; Sima 1996]. 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.

In particular, we focus on the triangular case of Equation (1), where both $A$ and $B$ are (upper) triangular matrices. The solution of the triangular case arises, e.g., as an intermediate subproblem in the Sylvester equation solver described in [Bartels and Stewart 1972]. 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 [Golub and Van Loan 1996].

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 high performance.

—A demonstration that high performance can be attained using the techniques here described.

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 already mentioned earlier paper on FLAME as well as a recent paper that gives theoretical insight into high-performance matrix multiplication algorithms [Gunnels et al. 2001]. Also, since the original submission of this paper we have refined the formal derivation approach. This more refined approach is described in detail in [Bientinesi et al. 2002]. In rewriting this paper, we have adopted the notation and steps of this latest paper.

This paper is structured as follows: In Section 2, we review traditional algorithms for the solution of the triangular Sylvester equation. A worksheet for the systematic derivation of linear algebra algorithms is discussed in Section 3. We use this worksheet to derive algorithms that are closely related to traditional algorithms as well as a more general family of algorithms in Section 4. In Section 5 we describe insights that we use to identify candidates from the family that are likely to yield high performance. Performance results on an Intel (R) Pentium (R) III processor are given in Section 6. A discussion of how the algorithms can be made practical is given in Section 7. A very brief discussion of experimental evidence of the numerical stability of the derived algorithms is given in Section 8. Concluding remarks follow in the final section.

## 2. TRADITIONAL SOLVERS FOR THE TRIANGULAR SYLVESTER EQUATION

Blocked algorithms usually obtain a higher performance in modern computers by rearranging the computations as possible in terms of matrix multiplication [Dongarra et al. 1990]. LAPACK [Anderson et al. 1995] 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 [Whaley and Dongarrra 1998], PHIPAC [Bilmes et al. 1997], and ITXGEMM [Gunnels et al. 2001].

In particular, 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, $B_{i,j}$, $i,j = 1,\ldots,n/b_n$. For simplicity, 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 [The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Mass, 01760 1993]. 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 [Kågström and Poromaa 1992].

```
for i = m/b_m : -1 : 1
    for j = 1 : n/b_n
        C_{i,j} = C_{i,j} - A_{i,i+1:m/b_m} X_{i,i+1:m/b_m,j}
        solve A_{i,i} X_{i,j} + X_{i,j} B_{j,j} = C_{i,j}
        C_{i,j+1:n/n_b} = C_{i,j+1:n/n_b} - X_{i,j} B_{j,j+1:n/n_b}
    end
end
```

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

| Step | Annotated Algorithm: $[D, E, F, \ldots] := \mathrm{op}(A, B, C, D, \ldots)$ |
|---|---|
| 1a | $\{P_{\mathrm{pre}}\}$ |
| 4 | **Partition**<br><br>**where** |
| 2 | $\{P_{\mathrm{inv}}\}$ |
| 3 | **while** $G$ **do** |
| 2,3 | $\{(P_{\mathrm{inv}}) \wedge (G)\}$ |
| 5a | **Repartition**<br><br><br>**where** |
| 6 | $\{Q_{\mathrm{before}}\}$ |
| 8 | $S_U$ |
| 7 | $\{Q_{\mathrm{after}}\}$ |
| 5b | **Continue with** |
| 2 | $\{P_{\mathrm{inv}}\}$ |
| | **enddo** |
| 2,3 | $\{(P_{\mathrm{inv}}) \wedge \neg (G)\}$ |
| 1b | $\{P_{\mathrm{post}}\}$ |

Fig. 2. Worksheet for developing linear algebra algorithms.

Recursive variants of these solvers have been recently developed in [Jonsson and Kågström 2001]. 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. A WORKSHEET FOR DERIVING LINEAR ALGEBRA ALGORITHMS

In Figure 2, we give a generic "worksheet" for deriving a large class of linear algebra algorithms. Expressions in curly-brackets denote predicates that describe the state of the various variables at the given point of the algorithm. For this paper, it suffices to realize that the statements between the assertions in the curly-brackets must be such that, at the indicated points in the algorithm, those assertions hold.

The generic linear algebra operation is given by $[D, E, F, \ldots] := \mathrm{op}(A, B, C, D, \ldots)$. Notice that some operands may be both input and output parameters. Constraints on these parameters, e.g. that describe structure or original contents, are given by the predicate $P_{\mathrm{pre}}$, the *precondition*. The *postcondition*, $P_{\mathrm{post}}$, is the predicate that describes the desired state upon completion of the algorithm.

We will require that the state given by the predicate $P_{\mathrm{inv}}$, the *loop-invariant*, is maintained at the top of the loop. Notice that thus $P_{\mathrm{inv}}$ must hold before the loop is entered, it must hold at the end of the loop so it will again hold at the top of the loop, and it will hold upon completion of the loop. This is indicated in Figure 2 at the various points where $P_{\mathrm{inv}}$ occurs in the assertions.

The *loop-guard*, $G$, is the condition under which the program remains in the loop. Thus, after the loop completes, $\neg G$ must hold. If $(P_{\mathrm{inv}} \wedge \neg G) \Rightarrow P_{\mathrm{post}}$ then we can conclude that the loop computes the desired result.

Since the loop-invariant must hold before the loop commences, the *initialization*, Step 4 in Figure 2, must have the property that starting in the state $P_{\mathrm{pre}}$, the initialization leaves us in a state where $P_{\mathrm{inv}}$ holds.

In order to make progress towards the condition under which the loop is completed, we will see that regions of the operands to be used and/or updated must be identified and added to regions that have already been updated in a consistent manner. It is this identification of submatrices and shifting of boundaries that occurs in Steps 5a and 5b in Figure 2.

The exposure of submatrices to be used and/or updated dictates the state $Q_{\mathrm{before}}$ before any updates have actually occurred. The update itself, $S_U$, must be such that the state $Q_{\mathrm{after}}$ holds. This state must be such that after the shifting of the boundaries the loop-invariant again holds at the bottom of the loop.

In the next section, we illustrate how the worksheet allows us to derive algorithms for the solution of (1).

## 4. DERIVATION OF ALGORITHMS FOR THE TRIANGULAR SYLVESTER EQUATION

Let us use the notation $C := X = \Omega(A, B, C)$ to denote the operation that overwrites matrix $C$ with the solution of (1). Let $\hat{C}$ denote the original contents of $C$ so that the precondition in Figure 2 becomes

$$C = \hat{C} \ \wedge \ \mathrm{UpTriang}(A) \ \wedge \ \mathrm{UpTriang}(B) \\ \wedge \ m(A) = n(A) = m(C) \ \wedge \ m(B) = n(B) = n(C) \ ;$$

here, $\mathrm{UpTriang}(Z)$ returns *true* if $Z$ is upper triangular, and $m(Z)$ and $n(Z)$ return, respectively, the row and column dimensions of $Z$. We now wish to compute

$$C = X = \Omega(A, B, \hat{C}), \tag{2}$$

the postcondition $P_{\mathrm{post}}$ in Figure 2.

**Note:** Throughout the rest of the paper, we will use the following notation:

| | |
|---|---|
| $\hat{C}$ | The original contents of matrix $C$ |
| $X$ | The solution of (1) |
| $C$ | The current contents of $C$ |

Table I. Feasible loop-invariants for the block-row oriented algorithms to solve the triangular Sylvester equation.

| Case | Loop-invariant | Resulting variant |
|---|---|---|
| R1 | $\left(\frac{C_T}{C_B}\right) = \left(\frac{\hat{C}_T}{\Omega(A_{BR}, B, \hat{C}_B)}\right)$ | Lazy |
| R2 | $\left(\frac{C_T}{C_B}\right) = \left(\frac{\hat{C}_T - A_{TR}\,\Omega(A_{BR}, B, \hat{C}_B)}{\Omega(A_{BR}, B, \hat{C}_B)}\right)$ | Eager |

First, we derive two block-row oriented (with respect to matrix $C$) solvers by partitioning only the first of the coefficient matrices, $A$. Analogous block-column oriented versions could be obtained by partitioning $B$ instead of $A$. Finally, we will investigate algorithms where both $A$ and $B$ are partitioned into four quadrants.

### 4.1 Block-row oriented solvers

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 $\{T\}$, $\{B\}$, $\{L\}$, and $\{R\}$ stand for top, bottom, left, and right, respectively. Accordingly, we next apply a conformal partition to $C$, $\hat{C}$, and $X$ by blocks of rows

$$C \rightarrow \left(\frac{C_T}{C_B}\right), \quad \hat{C} \rightarrow \left(\frac{\hat{C}_T}{\hat{C}_B}\right), \quad \text{and} \quad X \rightarrow \left(\frac{X_T}{X_B}\right),$$

where $C_B$, $\hat{C}_B$, and $X_B$ are $k_m \times n$ blocks.

With these partitionings, (1) can be rewritten as

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right)\left(\frac{X_T}{X_B}\right) + \left(\frac{X_T}{X_B}\right) B = \left(\frac{\hat{C}_T}{\hat{C}_B}\right).$$

Notice that the solution of this partitioned equation is given by

$$\left(\frac{X_T}{X_B}\right) = \Omega\left(\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), B, \left(\frac{\hat{C}_T}{\hat{C}_B}\right)\right) \tag{3}$$

$$= \left(\frac{\Omega(A_{TL}, B, \hat{C}_T - A_{TR}\,\Omega(A_{BR}, B, \hat{C}_B))}{\Omega(A_{BR}, B, \hat{C}_B)}\right). \tag{4}$$

To avoid the recomputation of an intermediate result, $X_B = \Omega(A_{BR}, B, \hat{C}_B)$, there are data dependencies which induce a strict order on the sequence of operations: Provided $C$ originally contains $\hat{C}$, first, $C_B := X_B = \Omega(A_{BR}, B, C_B)$ is solved, then the update $C_T := C_T - A_{TR}C_B$ is computed and, finally, $C_T := X_T = \Omega(A_{TL}, B, C_T)$ is solved.

While (4) gives all computations necessary to compute the solution, at an intermediate stage we would expect only some of these computations to have been computed. It is this insight that allows us to identify feasible loop-invariants, given in Table I. Notice that when any of these conditions is chosen as the loop-invariant, $P_{\text{inv}}$, the loop-guard $G : m(C_T) > 0$ has the property that $(P_{\text{inv}} \wedge \neg G) \Rightarrow P_{\text{post}}$.

Now, the initialization (Step 4 in Figure 2)

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

where $A_{BR}$ is $0 \times 0$ and $C_B$ and $\hat{C}_B$ have 0 rows has the property that, starting in a state where the precondition holds, it leaves the matrix $C$ in a state where $P_{\text{inv}}$ is *true*.

Next, we must make progress towards making $G$ *false*. In other words, we must derive a body of the loop that allows the computation of $C$ to proceed forward (up) by $b_m$ rows while ensuring that the loop-invariant is satisfied at the beginning of the next iteration. To derive these steps, we repartition matrices $A$, $C$, and $\hat{C}$ as

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right), \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array}\right), \left(\begin{array}{c} \hat{C}_T \\ \hline\hline \hat{C}_B \end{array}\right) \rightarrow \left(\begin{array}{c} \hat{C}_0 \\ \hline \hat{C}_1 \\ \hline\hline \hat{C}_2 \end{array}\right),$$

where $A_{11}$ is a $b_m \times b_m$ block and $C_1$ and $\hat{C}_1$ have $b_m$ rows. 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.

The double lines in these partitionings indicate how far the computation has progressed. The idea now is that progress is made by shifting various submatrices, from one side of the double lines to the other. Thus, at the bottom of the loop, we will continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right), \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) \leftarrow \left(\begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array}\right), \left(\begin{array}{c} \hat{C}_T \\ \hline\hline \hat{C}_B \end{array}\right) \leftarrow \left(\begin{array}{c} \hat{C}_0 \\ \hline\hline \hat{C}_1 \\ \hline \hat{C}_2 \end{array}\right).$$

4.1.1 *Lazy algorithm.* If we wish to maintain loop-invariant R1,

$$\left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right) = \left(\begin{array}{c} \hat{C}_T \\ \hline\hline \Omega(A_{BR}, B, \hat{C}_B) \end{array}\right),$$

in terms of these repartitioned matrices, the current contents of $C$ are given by

$$Q_{\text{before}} : \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array}\right) = \left(\begin{array}{c} \left(\begin{array}{c} \hat{C}_0 \\ \hline \hat{C}_1 \end{array}\right) \\ \hline\hline \Omega(A_{22}, B, \hat{C}_2) \end{array}\right). \tag{5}$$

As part of the body of the loop, the contents of $C$ must be updated so that

$$Q_{\text{after}} : \left(\begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array}\right) = \left(\begin{array}{c} \hat{C}_0 \\ \hline\hline \Omega\left(\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline 0 & A_{22} \end{array}\right), B, \left(\begin{array}{c} \hat{C}_1 \\ \hline \hat{C}_2 \end{array}\right)\right) \end{array}\right) \tag{6}$$

<table>
<tr><th>Step</th><th>Annotated Algorithm: $C := \Omega(A, B, C)$</th></tr>
<tr><td>1a</td><td>$\left\{ \begin{array}{l} C = \hat{C} \;\wedge\; \text{UpTriang}(A) \;\wedge\; \text{UpTriang}(B) \\ \wedge\, m(A) = n(A) = m(C) \;\wedge\; m(B) = n(B) = n(C) \end{array} \right\}$</td></tr>
<tr><td>4</td><td>Partition $A \rightarrow \left( \begin{array}{c||c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right), C \rightarrow \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right)$<br>where $A_{BR}$ is $0 \times 0$ and $C_B$ has 0 rows</td></tr>
<tr><td>2</td><td>$\left\{ \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) = \left( \begin{array}{c} \hat{C}_T \\ \hline\hline \Omega(A_{BR}, B, \hat{C}_B) \end{array} \right) \right\}$</td></tr>
<tr><td>3</td><td>while $m(C_T) > 0$ do</td></tr>
<tr><td>2,3</td><td>$\left\{ \left( \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) = \left( \begin{array}{c} \hat{C}_T \\ \hline\hline \Omega(A_{BR}, B, \hat{C}_B) \end{array} \right) \right) \wedge (m(C_T) > 0) \right\}$</td></tr>
<tr><td>5a</td><td>Determine block size $b$<br>Repartition<br>$\left( \begin{array}{c||c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right) \rightarrow \left( \begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array} \right), \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) \rightarrow \left( \begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array} \right)$<br>where $A_{11}$ is $b_m \times b_m$ and $C_1$ has $b_m$ rows</td></tr>
<tr><td>6</td><td>$\left\{ \left( \begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array} \right) = \left( \begin{array}{c} \left( \begin{array}{c} \hat{C}_0 \\ \hline \hat{C}_1 \end{array} \right) \\ \hline\hline \Omega(A_{22}, B, \hat{C}_2) \end{array} \right) \right\}$</td></tr>
<tr><td>8</td><td>$C_1 := C_1 - A_{12} C_2$<br>$C_1 := \Omega(A_{11}, B, C_1)$</td></tr>
<tr><td>7</td><td>$\left\{ \left( \begin{array}{c} \hat{C}_0 \\ \hline\hline \left( \begin{array}{c} \Omega(A_{11}, B, \hat{C}_1 - A_{12}\Omega(A_{22}, B, \hat{C}_2)) \\ \hline \Omega(A_{22}, B, \hat{C}_2)) \end{array} \right) \end{array} \right) \right\}$</td></tr>
<tr><td>5b</td><td>Continue with<br>$\left( \begin{array}{c||c} A_{TL} & A_{TR} \\ \hline 0 & A_{BR} \end{array} \right) \leftarrow \left( \begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array} \right), \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) \leftarrow \left( \begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array} \right)$</td></tr>
<tr><td>2</td><td>$\left\{ \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) = \left( \begin{array}{c} \hat{C}_T \\ \hline\hline \Omega(A_{BR}, B, \hat{C}_B) \end{array} \right) \right\}$</td></tr>
<tr><td></td><td>enddo</td></tr>
<tr><td>2,3</td><td>$\left\{ \left( \left( \begin{array}{c} C_T \\ \hline\hline C_B \end{array} \right) = \left( \begin{array}{c} \hat{C}_T \\ \hline\hline \Omega(A_{BR}, B, \hat{C}_B) \end{array} \right) \right) \wedge \neg\, (m(C_T) > 0) \right\}$</td></tr>
<tr><td>1b</td><td>$\left\{ C = X = \Omega(A, B, \hat{C}) \right\}$</td></tr>
</table>

Fig. 3. Worksheet for the lazy block-row algorithm to solve the triagular Sylvester equation.

$$= \left( \begin{array}{c} \hat{C}_0 \\ \hline\hline \left( \begin{array}{c} \Omega(A_{11}, B, \hat{C}_1 - A_{12}\Omega(A_{22}, B, \hat{C}_2)) \\ \hline \Omega(A_{22}, B, \hat{C}_2) \end{array} \right) \end{array} \right). \qquad (7)$$

The question now is what the statements, $S_U$, are that allow the computation to move forward while maintaining the indicated loop-invariant. Comparing (5) and (7), we conclude that we need only to perform the operations:

$$C_1 := C_1 - A_{12} C_2 \,,$$

**Partition** $A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), C \rightarrow \left(\begin{array}{c} C_T \\ \hline\hline C_B \end{array}\right)$
  **where** $A_{BR}$ is $0 \times 0$ and $C_B$ has 0 rows
**while** $m(C_T) > 0$ **do**
  **Determine block size** $b$
  **Repartition**

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

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

| Lazy variant: | Eager variant: |
|---|---|
| $C_1 := C_1 - A_{12}C_2$ | $C_1 := \Omega(A_{11}, B, C_1)$ |
| $C_1 := \Omega(A_{11}, B, C_1)$ | $C_0 := C_0 - A_{01}C_1$ |

  **Continue with**

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

**enddo**

Fig. 4. 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:* |
|---|---|
| $C_{i,:} = C_{i,:} - A_{i,i+1:m/b_m} C_{i+1:m/b_m,:}$ | `solve` $A_{i,i}C_{i,:} + C_{i,:}B = C_{i,:}$ |
| `solve` $A_{i,i}C_{i,:} + C_{i,:}B = C_{i,:}$ | $C_{1:i-1,:} = C_{1:i-1,:} - A_{1:i-1,i}C_{i,:}$ |

`end`

Fig. 5. Lazy and eager traditional row-block oriented triangular Sylvester equation solvers.

$$C_1 := \Omega(A_{11}, B, C_1)\,.$$

The completed worksheet is stated in Figure 3. By recognizing that $\hat{C}$ is only introduced for the sake of the assertions in curly-brackets, the final algorithm is given in Figure 4. We classify the resulting algorithm as "lazy", in the sense that at a given stage, as little as possible of $C$ has been updated while allowing progress toward the solution to be made.

For those more comfortable with traditional algorithms, this is equivalent to the solver in Figure 5 (left). The lazy row-block oriented algorithms just presented, in the FLAME and the traditional formulations, are special cases of the algorithm in Figure 1, with $b_n = n$.

4.1.2 *Eager algorithm.* If we wish to maintain the loop-invariant R2 in terms of the repartitioned matrices the current contents of $C$ are instead given by

$$Q_{\text{before}}: \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline\hline C_2 \end{array}\right) = \left(\begin{array}{c} \left(\begin{array}{c} \hat{C}_0 - A_{02}\Omega(A_{22}, B, \hat{C}_2) \\ \hline \hat{C}_1 - A_{12}\Omega(A_{22}, B, \hat{C}_2) \end{array}\right) \\ \hline\hline \Omega(A_{22}, B, \hat{C}_2) \end{array}\right). \tag{8}$$

Now the contents of $C$ must be updated so that

$$Q_{\text{after}}: \left(\begin{array}{c} C_0 \\ \hline\hline C_1 \\ \hline C_2 \end{array}\right) = \left(\begin{array}{c} \hat{C}_0 - \left( A_{01} \,|\, A_{02} \right) \Omega\left(\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline 0 & A_{22} \end{array}\right), B, \left(\begin{array}{c} \hat{C}_1 \\ \hline \hat{C}_2 \end{array}\right)\right) \\ \hline\hline \Omega\left(\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline 0 & A_{22} \end{array}\right), B, \left(\begin{array}{c} \hat{C}_1 \\ \hline \hat{C}_2 \end{array}\right)\right) \end{array}\right) \tag{9}$$

$$= \left(\begin{array}{c} \hat{C}_0 - A_{12}\Omega(A_{11}, B, \hat{C}_1 - A_{12}\Omega(A_{22}, B, \hat{C}_2)) - A_{02}\Omega(A_{22}, B, \hat{C}_2) \\ \hline\hline \left(\begin{array}{c} \Omega(A_{11}, B, \hat{C}_1 - A_{12}\Omega(A_{22}, B, \hat{C}_2)) \\ \hline \Omega(A_{22}, B, \hat{C}_2) \end{array}\right) \end{array}\right) \tag{10}$$

Comparing (8) and (10), we conclude that we need only to perform the operations:

$$\begin{aligned} C_1 &:= \Omega(A_{11}, B, C_1), \\ C_0 &:= C_0 - A_{01}C_1. \end{aligned}$$

The algorithm is stated in Figure 4 and is equivalent to the traditional solver in Figure 5 (right). We classify the resulting algorithm as "eager", in the sense that at a given stage, as much as possible of $C$ has been updated without computing the final answer.

4.1.3 *Proving correctness and cost.* The following theorems prove 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 algorithms in Figure 4 overwrite matrix $C$ with the solution of the triangular Sylvester equation $AX+XB = C$.*

*Proof. For the lazy algorithm, the proof follows from the fact that the algorithm was derived so that the loop terminates and the assertions in Figure 3 are* true. *A similar worksheet can be given for the eager algorithm.* □

THEOREM 2. *The lazy and eager block-row oriented triangular Sylvester equation solvers in Figure 4 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 4 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:*

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

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

| Operation | Cost | |
|---|---|---|
| | Lazy variant | Eager variant |
| $C_1 := C_1 - A_{12}C_2$ | $\sum_{k=0}^{m/b_m-1} 2b_m k_m n \approx m^2 n$ | |
| $C_1 := \Omega(A_{11}, B, C_1)$ | $\sum_{k=0}^{m/b_m-1} (b_m^2 n + b_m n^2) \approx mn^2$ | $\sum_{k=0}^{m/b_m-1} (b_m^2 n + b_m n^2) \approx mn^2$ |
| $C_0 := C_0 - A_{01}C_1$ | | $\sum_{k=0}^{m/b_m-1} 2\bar{m} b_m n \approx m^2 n$ |
| Total | $m^2 n + mn^2$ | $m^2 n + mn^2$ |

*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*

$$\begin{array}{ll} C_1 := C_1 - A_{12}C_2 & 2b_m k_m n \\ C_1 := \Omega(A_{11}, B, C_1) & b_m^2 n + b_m n^2 \\ C_0 := C_0 - A_{01}C_1 & 2\bar{m} b_m n \end{array}$$

*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 II reports the cost of these three operations and the overall cost of the algorithms, proving the theorem.*

□

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.

4.1.4 *Implementation.* The FLAME library allows code to mirror the derived algorithms, thus largely inheriting the proven correctness. Implementations of the lazy and eager algorithms using FLAME are given in Figure 6.

## 4.2 Block-column oriented solvers

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

## 4.3 A Family of Blocked Algorithms

We now show that a partitioning of both coefficient matrices leads to a family of sixteen different blocked algorithms.

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

```
int FLA_Syl_Block_Row( int variant, FLA_Obj A, FLA_Obj B, FLA_Obj C, int bm_alg )
{
  // 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 ){
    bm = min( FLA_Obj_length( CT ), bm_alg );

    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 );
    /* ************************************************************** */

    if ( variant == LAZY )                                  /* C1 <- C1 - A12 C2 */
      FLA_Gemm( FLA_NO_TRANSPOSE, FLA_NO_TRANSPOSE, MINUS_ONE, A12, C2, ONE, C1 );

    FLA_Syl_level2( A11, B, C1 );                    /* C1 <- Omega( A11, B, C1 ) */

    if ( variant == EAGER )                                 /* C0 <- C0 - A01 C1 */
      FLA_Gemm( FLA_NO_TRANSPOSE, FLA_NO_TRANSPOSE, MINUS_ONE, A01, C1, ONE, C0 );

    /* *************************************************************** */
    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 );
  }
}
```

Fig. 6. Block-row oriented triangular Sylvester equation solvers implemented using FLAME.

apply a conformal partition to $C$, $\hat{C}$, and $X$:

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

where $C_{BL}$, $\hat{C}_{BL}$, and $X_{BL}$ are $k_m \times k_n$ blocks.

Now, (1) becomes

$$\left( \begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array} \right) \left( \begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array} \right) + \left( \begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array} \right) \left( \begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array} \right) = \left( \begin{array}{c||c} \hat{C}_{TL} & \hat{C}_{TR} \\ \hline\hline \hat{C}_{BL} & \hat{C}_{BR} \end{array} \right)$$

$$\begin{tikzcd}[column sep=small]
 & C_{BL} := X_{BL} = \Omega(A_{BR}, B_{TL}, \hat{C}_{BL}) \arrow[dl] \arrow[dr] & \\
C_{TL} := \hat{C}_{TL} - A_{TR}X_{BL} \arrow[d] & & C_{BR} := \hat{C}_{BR} - X_{BL}B_{TR} \arrow[d] \\
C_{TL} := X_{TL} = \Omega(A_{TL}, B_{TL}, C_{TL}) \arrow[d] & & C_{BR} := X_{BR} = \Omega(A_{BR}, B_{BR}, C_{BR}) \arrow[d] \\
C_{TR} := C_{TR} - X_{TL}B_{TR} \arrow[dr] & & C_{TR} := C_{TR} - A_{TR}X_{BR} \arrow[dl] \\
 & C_{TR} := X_{TR} = \Omega(A_{TL}, B_{BR}, C_{TR}) &
\end{tikzcd}$$

Fig. 7. Data dependencies for the partitioned triangular Sylvester matrix equation.

and the solution, in partitioned form, is given by

$$\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) = \Omega\left(\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right), \left(\begin{array}{c||c} \hat{C}_{TL} & \hat{C}_{TR} \\ \hline\hline \hat{C}_{BL} & \hat{C}_{BR} \end{array}\right)\right) \tag{11}$$

$$= \left(\begin{array}{c||c} \Omega(A_{TL}, B_{TL}, \hat{C}_{TL} - A_{TR}X_{BL}) & \Omega(A_{TL}, B_{BR}, \hat{C}_{TR} - A_{TR}X_{BR} - X_{TL}B_{TR}) \\ \hline\hline \Omega(A_{BR}, B_{TL}, \hat{C}_{BL}) & \Omega(A_{BR}, B_{BR}, \hat{C}_{TR} - X_{BL}B_{TR}) \end{array}\right). \tag{12}$$

The dependencies among the operations induce a certain order: the first operation that must be performed is solving $C_{BL} := X_{BL} = \Omega(A_{BR}, B_{TL}, \hat{C}_{BL})$; after that, only the updates $C_{TL} := \hat{C}_{TL} - A_{TR}X_{BL}$ or $C_{BR} := \hat{C}_{BR} - X_{BL}B_{TR}$ are possible, and so on. Figure 7 shows graphically these data dependencies. Due to the dependencies, there are only sixteen valid loop-invariants, as summarized in Table III. Notice that we label six of these cases as duals, since they involve analogous updates.

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

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

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

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

where $C_{11}$ is a $b_m \times b_n$ block and consider analogous repartitionings of $\hat{C}$ and $X$. Notice that, again, the double lines mark how far (the boundaries of) the computation has progressed. For the blocked algorithms, the solvers march through matrices $C$, $\hat{C}$, and $X$ from the bottom-left corner to the top-right one, moving $b_m$ rows and $b_n$ columns at each iteration. In $A$, the double lines move towards the top-left corner, while in $B$ the direction is towards the bottom-right corner. (If $A$

Table III. Valid loop-invariants for the blocked algorithms to solve the triangular Sylvester equation.

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

and/or $B$ were lower triangular matrices, all other possibilities in the direction of the computation would be observed.)

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

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

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

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

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

with $\hat C$ and $X$ repartitioned as $C$. With this new repartitioning, we wish the

Algorithm 1. $C \leftarrow X$, *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** $0 \times 0$

**while** $C_{TR}$ **is not** $0 \times 0$ **do**

**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:* |
|---|---|---|
| $C_{10} := C_{10} - A_{12}C_{20}$ | | |
| $C_{10} := \Omega(A_{11}, B_{00}, C_{10})$ | $C_{10} := \Omega(A_{11}, B_{00}, C_{10})$ | $C_{10} := \Omega(A_{11}, B_{00}, C_{10})$ |
| $C_{11} := C_{11} - C_{10}B_{01}$ | $C_{11} := C_{11} - C_{10}B_{01}$ | $C_{11} := C_{11} - C_{10}B_{01}$ |
| $C_{21} := C_{21} - C_{20}B_{01}$ | $C_{21} := C_{21} - C_{20}B_{01}$ | $C_{21} := \Omega(A_{22}, B_{11}, C_{21})$ |
| $C_{21} := \Omega(A_{22}, B_{11}, C_{21})$ | $C_{21} := \Omega(A_{22}, B_{11}, C_{21})$ | |
| $C_{11} := C_{11} - A_{12}C_{21}$ | $C_{11} := C_{11} - A_{12}C_{21}$ | $C_{11} := C_{11} - A_{12}C_{21}$ |
| $C_{11} := \Omega(A_{11}, B_{11}, C_{11})$ | $C_{11} := \Omega(A_{11}, B_{11}, C_{11})$ | $C_{11} := \Omega(A_{11}, B_{11}, C_{11})$ |
| | $C_{00} := C_{00} - A_{01}C_{10}$ | $C_{00} \leftarrow C_{00} - A_{01}C_{10}$ |
| | $C_{01} := C_{01} - A_{01}C_{11}$ | $C_{01} \leftarrow C_{01} - A_{01}C_{11}$ |
| | $C_{01} := C_{01} - A_{02}C_{21}$ | $C_{01} \leftarrow C_{01} - A_{02}C_{21}$ |
| | | $C_{12} \leftarrow C_{12} - C_{10}B_{02}$ |
| | | $C_{12} \leftarrow C_{12} - C_{11}B_{12}$ |
| | | $C_{22} \leftarrow C_{22} - C_{21}B_{12}$ |

**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**

Fig. 8. Blocked triangular Sylvester equation solvers derived from C1, C2, and C3.

contents of $C$ to become

$$Q_{\text{after}} : \left( \begin{array}{c||c} C_{TL} - A_{TR}X_{BL} & C_{TR} \\ \hline\hline X_{BL} & C_{BR} \end{array} \right)$$

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

Therefore, in order to move the computation forward in case C2 it can be easily shown that the operations shown in Figure 8 (center) must be performed (notice that $\hat{C}$ and $X$ are only introduced for the sake of the assertions in curly-brackets). We leave it as an exercise to the reader to derive the other variants.

Theorem 3. *The algorithms in Figure 8 overwrite matrix $C$ with the solution* *of the triangular Sylvester equation $AX + XB = C$.*

*Proof.* *Generating the worksheets for the blocked algorithms would yield a proof 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 8 requires $m^2 n + mn^2$ 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 8 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:*

$$\begin{array}{ccc|c} \overbrace{A_{00}}^{\bar m} & \overbrace{A_{01}}^{b_m} & \overbrace{A_{02}}^{k_m} & \}\bar m \\ \hline 0 & A_{11} & A_{12} & \}b_m \\ \hline\hline 0 & 0 & A_{22} & \}k_m \end{array} \qquad \begin{array}{ccc|c} \overbrace{B_{00}}^{k_n} & \overbrace{B_{01}}^{b_n} & \overbrace{B_{02}}^{\bar n} & \}k_n \\ \hline\hline 0 & B_{11} & B_{12} & \}b_n \\ \hline 0 & 0 & B_{22} & \}\bar n \end{array} \qquad \begin{array}{ccc|c} \overbrace{C_{00}}^{k_n} & \overbrace{C_{01}}^{b_n} & \overbrace{C_{02}}^{\bar n} & \}\bar m \\ \hline C_{10} & C_{11} & C_{12} & \}b_m \\ \hline\hline C_{20} & C_{21} & C_{22} & \}k_m \end{array}$$

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

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

$$\begin{array}{ll} C_{10} := \Omega(A_{11}, B_{00}, C_{10}) & b_m^2 k_n + b_m k_n^2 \\ C_{21} := \Omega(A_{22}, B_{11}, C_{21} - C_{20}B_{01}) & 2b_n k_m k_n + b_n^2 k_m + b_n k_m^2 \\ C_{11} := \Omega(A_{11}, B_{11}, C_{11} - C_{10}B_{01} - A_{12}C_{21}) & 2b_m b_n k_n + 2b_m b_n k_m + b_m^2 b_n + b_m b_n^2 \\ C_{00} := C_{00} - A_{01}C_{10} & 2b_m k_n \bar m \\ C_{01} := C_{01} - A_{01}C_{11} - A_{02}C_{21} & 2b_m b_n \bar m + 2b_n k_m \bar m \end{array}$$

*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 $k_n = kb_n$, and some algebra proves the theorem.* □

As we would hope, the blocked algorithm requires the same computational cost as the block-row solvers. 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.

Table IV. Factors affecting the performance of the matrix multiplication.

| Shape | Performance |
|---|---|
| two dimensions large, one dimension equal to $b_2$. | Best |
| one dimension large, two dimensions equal to $b_2$. | $\uparrow$ |
| two dimensions equal to $b_2$, one dimension equal to $b_1$. | $\downarrow$ |
| one dimension equal to $b_2$, two dimensions equal to $b_1$. | Good |
| all dimensions $\leq b_1$ | Worse |
| one or more dimensions equal to 1 | Worst |

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 (DGEMM) 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 [Gunnels et al. 2001], it is shown that, for architectures with two levels of cache memory, there are two block sizes that influence the performance of DGEMM: $b_1$ and $b_2$, which are related to the size of the L1 and L2 caches, respectively. In Table IV we show how the three matrix dimensions, $m$, $n$, and $k$ affect performance of DGEMM. 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 DGEMM 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 subsection 4.1 we mentioned that the block-row oriented algorithms spend approximately half of the computation in these subproblems. By symmetry, 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 subproblems.

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 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 such case, approximately a third of the computation is spent in the subproblems. We can conclude that we should pursue the use of square block sizes.

Let us revisit the observations made regarding the matrix-matrix multiply. Table V gives two heuristics for choosing different blocked and/or block-row and block-column when targeting different levels of the memory hierarchy. Consider the top-level blocking. If one takes $b_m = b_n = b_2$ (as in Heuristic 1) 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 $b_m = b_2$ (as in Heuristic 2),

Table V. Two heuristics for choosing algorithms and block sizes.

| Case $m$ | Case $n$ | Strategy: Heuristic 1 | Strategy: Heuristic 2 | Comment: dims. passed to DGEMM |
|---|---|---|---|---|
| large | large | blocked ($b_m = b_n = b_2$) | block-row ($b = b_2$) | two large/one equal to $b_2$. |
| $b_2$ | large | block-column ($b = b_2$) | | one large/two equal to $b_2$. |
| large | $b_2$ | block-row ($b = b_2$) | | one large/two equal to $b_2$. |
| $b_2$ | $b_2$ | blocked ($b_m = b_n = b_1$) | block-row ($b = b_1$) | two equal to $b_2$/one equal to $b_1$. |
| $b_1$ | $b_2$ | block-column ($b = b_1$) | | one equal to $b_2$/two equal to $b_1$. |
| $b_1$ | $b_1$ | any | | blocking is less important. |

Table VI. Table that illustrates that by picking the block sizes appropriately, some of the blocked algorithms become block-row or block-column algorithms.

| Resulting algorithm | Cases: $b_m = m$ | Cases: $b_n = n$ |
|---|---|---|
| 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 |

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 DGEMM are made. Thus, *in practice*, we can expect the calls to DGEMM employed by the block-row or block-column algorithms to attain higher performance than the calls employed by the blocked algorithms when $b_m = b_n = b_2$.

Heuristic 2 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. As indicated in Table VI, 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. 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 to 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 subproblems that remain at the lowest level of the recursion. For example, we can employ the same algorithm to reduce the subproblem 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 subproblems 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.

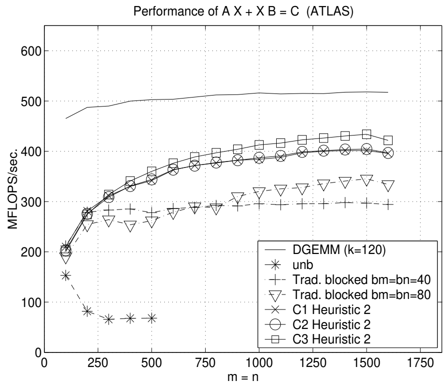
We report performance on an Intel (R) Pentium (R) III (650 MHz) processor with a 16 Kbyte L1 data cache and a 256 Kbyte L2 cache running RedHat Linux 7.1. All computations were performed in 64-bit (double precision) arithmetic, and the same options were used when compiling the different implementations. For all experiments, the implementations were linked to an implementation of the Basic Linear Algebra Subprograms (BLAS) [Lawson et al. 1979; Dongarra et al. 1988; Dongarra et al. 1990] provided by ATLAS (Release R3.2). This library provides high-performance implementations of commonly encountered matrix operations. For some experiments, the matrix multiplication kernel (DGEMM) provided by ATLAS was replaced by our own high-performance implementation, ITXGEMM [Gunnels et al. 2001], while the rest of the BLAS were provided by ATLAS.

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 subproblems are computed using the unblocked (`unb`) solver. Although we implemented all blocked, block-row, and block-column algorithms, we only present results for C1, C2, and C3. The remaining blocked algorithms obtained performance that was virtually identical to C1, C2, and C3. Also, we already argued that the block-row and block-column algorithms are special cases of C1, C2, C3, and C11.

Figure 9 (LEFT) reports the performance for these algorithms using DGEMM from ATLAS R3.2. For reference, we also include the performance of the DGEMM routine for a matrix multiplication with $k = b_2$. We only report the results using the second heuristic described in the previous section, with block sizes $120 \times n$, $120 \times 120$, $40 \times 120$, $40 \times 40$, $20 \times 40$, and $20 \times 20$, for the different levels of the recursion. It is interesting to note that algorithm C3 performs somewhat better than C1 and C2.

Experts in the field will appreciate the fact that C3 performs most of its computation in the updates $C_{00} := C_{00} - A_{01}C_{10}$ and $C_{22} := C_{22} - C_{21}B_{12}$, which are both rank-k updates. Typical implementations of DGEMM are tuned to perform best for that particular case of matrix multiplication. By contrast, C1 and C2 perform some or most of their computation in updates that involve matrices with small “m” or



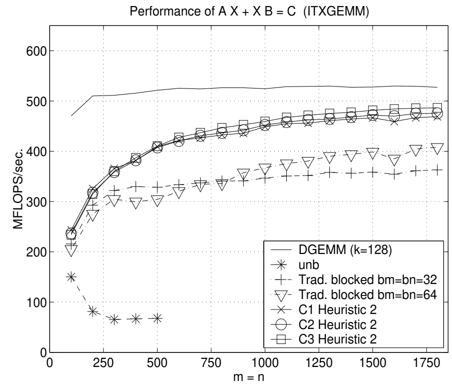




Fig. 9. Performance of our triangular Sylvester equation solvers using DGEMM from ATLAS R3.2 (LEFT) and ITXGEMM (RIGHT).

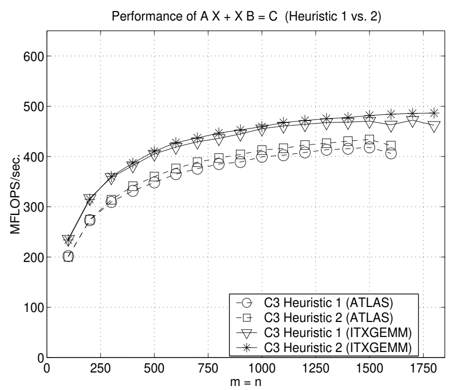



Fig. 10. Performance of the triangular Sylvester equation solvers using heuristic 1 and heuristic 2.

"n" dimension, performance of which is often not as highly tuned.

In Figure 9 (RIGHT) we report performance from the same experiments as reported in Figure 9 using DGEMM from ITXGEMM instead. For ITXGEMM, block sizes $128 \times n$, $128 \times 128$, $64 \times 128$, $64 \times 64$, $16 \times 64$, and $16 \times 16$, for the different levels of the recursion, yield high performance. Since this implementation of matrix multiplication is highly tuned for all matrix sizes, differences in performance of C1, C2, and C3 is less noticable. In addition, a higher percentage of the performance of matrix multiplication is achieved.

Figure 10 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 noticeably from the larger matrix sizes involved in the calls to DGEMM in Heuristic 2.

Although not reported here, our solvers, when linked to ATLAS R3.2, obtain performance similar to that reported in [Jonsson and Kågström 2001] when we adjust for the different clock rate of the processor on which they performed their experiments.

## 7. APPLICATION TO PRACTICAL IMPLEMENTATIONS

This paper is intended to demonstrate the application of the FLAME approach to a more complex operation. However, we now show how the insights gained in the simpler setting where matrices $A$ and $B$ are triangular matrices can be easily extended to the more complex case that arises in practice.

### 7.1 Solution of the general Sylvester equation

Let us start by considering the general case as it typically occurs in practice, for example in control theory:

$$\bar{A}\bar{X} + \bar{X}\bar{B} = \bar{C} \tag{13}$$

where $\bar{A}$ and $\bar{B}$ are general, real valued, square matrices. A typical first step is to first compute real Schur decompositions of matrices $\bar{A}$ and $\bar{B}$:

$$\bar{A} \rightarrow Q_A A Q_A^T \quad \text{and} \quad \bar{B} \rightarrow Q_B B Q_B^T \tag{14}$$

where $Q_A$ and $Q_B$ are unitary matrices and $A$ and $B$ are *quasi* upper triangular (block triangular with $1 \times 1$ and $2 \times 2$ blocks on the diagonal). Then

$$(Q_A^T \bar{A} Q_A)(Q_A^T \bar{X} Q_B) + (Q_A^T \bar{X} Q_B) Q_B^T \bar{B} Q_B = (Q_A^T \bar{C} Q_B)$$

or

$$AX + XB = C \tag{15}$$

where $X = Q_A^T \bar{X} Q_B$ and $C = Q_A^T \bar{C} Q_B$. Thus, the steps for computing the solution to (13) become:

(1) Compute the real Schur decompositions. Approximate cost: $30(m^3+n^3)$ flops[1].
(2) Compute $C = Q_A^T \bar{C} Q_B$. Approximate cost: $2m^3 + 2n^3$ flops.
(3) Solve (15). Approximate cost: $m^2 n + mn^2$ flops.
(4) Compute $\bar{X} = Q_A X Q_B^T$. Approximate cost: $2m^3 + 2n^3$ flops.

This then produces the desired solution of the original problem stated in (13).

The first observation is that the cost of the solution of the triangular Sylvester equation is only a small part of the overall cost.

### 7.2 Solution of the quasi triangular Sylvester equation

The second observation is that what is really needed is a family of algorithms for the solution of the quasi-triangular Sylvester equation. We now discuss how the algorithms presented in the previous sections must be adjusted for this case.

(1) **$A$ is at most $2 \times 2$ and $B$ is $1 \times 1$:** In this case (15) is equivalent to $(A+\beta I)x = c$, where $B = \beta$ is a scalar and $x$ and $c$ are vectors. Since $A$ is at most $2 \times 2$ this equation can be solved via Gaussian Elimination with partial pivoting.

[1]Part of this computation is an iteration that converges to the decomposition. Since the rate of convergence depends on the spectrum of the matrices, this operation count varies depending on the input matrices.

(2) $A$ **is at most** $2 \times 2$ **and** $B$ **is** $2 \times 2$**:** In this case (15) is given by

$$A \begin{pmatrix} x_1 & x_2 \end{pmatrix} + \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \end{pmatrix}$$

where $\beta_{ij}$ is a scalar and $x_j$ and $c_j$ are vectors. This problem can be restated as

$$\left(\begin{array}{c|c} A + \beta_{11} I & \beta_{21} I \\ \hline \beta_{12} I & A + \beta_{22} I \end{array}\right) = \left(\begin{array}{c} c_1 \\ \hline c_2 \end{array}\right).$$

Since this system is at most $4 \times 4$, we can again use Gaussian Elimination with partial pivoting.

(3) $A$ **is upper triangular and** $B$ **is** $1 \times 1$**:** In this case (15) is again equivalent to $(A + \beta I)x = c$. Since $A$ is upper triangular, so is $A + \beta I$ and therefore any algorithm for the solution of a triangular system can be used.

(4) $A$ **is quasi-triangular and** $B$ **is** $1 \times 1$**:** For this case, we can apply any of the algorithms derived in Section 4.1 with the constraint that the block size $b_m$ should always be chosen so that $A_{11}$ is either upper triangular or $2 \times 2$. If $A_{11}$ is $2 \times 2$ the subproblem $C_{11} := \Omega(A_{11}, B, C_{11})$ can then be computed using the approach in case 1. If it is upper triangular, the approach in case 3 above can be employed.

(5) $A$ **is quasi-triangular and** $B$ **is** $2 \times 2$**:** For this case, we can apply any of the algorithms derived in Section 4.1 with the constraint that the block size $b_m$ should always be chosen so that $A_{11}$ is either $1 \times 1$ or $2 \times 2$. The subproblem $C_{11} := \Omega(A_{11}, B, C_{11})$ can then be computed using the approach in case 2 above.

(6) $A$ **and** $B$ **are both quasi-triangular:** If both $A$ and $B$ are quasi-triangular then any of the blocked algorithms discussed in previous sections can be applied provided that
   - The block sizes $b_m$ and/or $b_n$ are chosen so that $A_{10}$, $A_{20}$, $A_{21}$, $B_{10}$, $B_{20}$, and $B_{21}$ all only contain zeroes.
   - Eventually an algorithm is called for the smaller subproblems with $1 \times 1$ or $2 \times 2$ matrix $B_{11}$.

With this, we have arrived at a practical implementation for the quasi-triangular Sylvester equation.

## 8. STABILITY EXPERIMENTS

The stability analysis of unblocked algorithms has been well studied. For example, a thorough treatment is given in [Higham 2002]. In this section, we discuss experimental data that provides some evidence that the derived blocked algorithms exhibit stability similar to that of the unblocked algorithms. Additional comments about the need for systematic stability analyses are given in the conclusion of the paper.

All stability experiments in this section were run on an Intel (R) Pentium III platform using Matlab 6.0 (machine precision $\varepsilon \approx 2.2 \times 10^{-16}$). Implementations of all the codes and a special version of the FLAME API (for Matlab) were developed for that purpose.

Table VII. Numerical performance of the Sylvester solvers.

| $p$ | $\|X-\hat{X}\|_\infty/\|\hat{X}\|_\infty$ | | | |
|---|---|---|---|---|
| | `unb` | C1 | C2 | C3 |
| 10 | 1.2965e-10 | 1.4104e-10 | 1.4104e-10 | 1.4104e-10 |
| 20 | 2.3366e-07 | 2.3365e-07 | 2.3365e-07 | 2.3365e-07 |
| 30 | 8.7028e-04 | 8.7028e-04 | 8.7028e-04 | 8.7028e-04 |
| 40 | 8.4664e-01 | 8.4666e-01 | 8.4666e-01 | 8.4666e-01 |

In order to evaluate the numerical properties of our solvers, we use an example of a generalized Sylvester equation

$$\hat{A}X\hat{B} + \hat{C}X\hat{D} = \hat{E}, \tag{16}$$

from [Gardiner et al. 1992]. In that example

$$\begin{aligned}
\hat{A} &= \mathrm{diag}(1,2,\ldots,m) + U_m,\\
\hat{B} &= I_n + 2^{-p}U_n^T\\
\hat{C} &= I_m + 2^{-p}U_m^T,\\
\hat{D} &= 2^{-p}I_n - \mathrm{diag}(n,n-1,\ldots,1) + U_m,
\end{aligned}$$

where $p$ is a parameter and $U_k$ is a $k\times k$ matrix with unit entries below the diagonal and all other entries zero. As $p$ increases, the system approaches singularity and the numerical condition of (16) gets worse. The problem is then transformed into an standard Sylvester Equation (1) by solving the systems $A = \hat{C}^{-1}\hat{A}$ and $B = \hat{D}^T\hat{B}^{-T}$. The right-hand side matrix $C$ is set to be an $m\times n$ matrix so that the true solution matrix $X$ has all unit entries. Matrices $A$ and $B$ are then reduced to real Schur form using the QR algorithm, resulting in quasi-triangular matrices with $1\times 1$ and $2\times 2$ diagonal blocks. The right-hand side matrix $C$ is updated with the corresponding orthogonal matrices computed by the QR algorithm.

Table VII reports the relative error

$$\|X-\tilde{X}\|_\infty/\|\hat{X}\|_\infty$$

of the solution $\tilde{X}$ computed by means of a traditional non-blocked solver (`unb`) and our variants C1, C2, and C3 for a problem of size $m = n = 128$. The remaining blocked algorithms obtained numerical performances that are very similar to those of C1, C2, and C3. At the lowest level, all algorithms dealt with tiny Sylvester equations of size $1\times 1$, $1\times 2$, $2\times 1$, and $2\times 2$, transforming these into an equivalent small linear system, and using Gaussian elimination with partial pivoting to solve these small subproblems. Our blocked variants employed a square block size $b_m = b_n = 64$. At deeper levels of the recursion $b_m$ and $b_n$ are set to 1 or 2, depending on whether a $2\times 2$ block is encountered in the diagonal of the quasi-triangular matrices $A$ and/or $B$. No significant differences were found when we used non-square block sizes, smaller block sizes at deeper levels of the recursion, etc.

The conclusion of our experiments is that it appears that the derived blocked algorithms exhibit numerical stability that is similar to that of the unblocked algorithms.

## 9. 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.

—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.

Clearly, this paper raises as many questions as it solves. For example, given that a large number of algorithms are systematically derived, the question of the numerical stability of the resulting algorithms must be addressed. While for some operations, for the triangular Sylvester equation whether or not all algorithms are equally stable is not immediately obvious. Thus, a systematic approach to deriving stability results would be highly desirable. We intend to study this question in the future.

### 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. We are particularly grateful to G.W. (Pete) Stewart for his valuable feedback on the original manuscript.

### Additional Information

For additional information on FLAME please visit

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

## REFERENCES

ALIEV, F. AND LARIN, V. 1998. *Optimization of Linear Control Systems: Analytical Methods and Computational Algorithms.* Stability and Control: Theory, Methods and Applications, vol. 8. Gordon and Breach.

ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1995. *LAPACK Users' Guide*, second ed. SIAM, Philadelphia, PA.

BARTELS, R. AND STEWART, G. 1972. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Comm. ACM 15*, 820–826.

BIENTINESI, P., GUNNELS, J. A., MYERS, M. E., QUINTANA-ORÍ, E. S., AND VAN DE GEIJN, R. A. 2002. The science of deriving dense linear algebra algorithms. Tech. Rep. CS-TR-02-53, Department of Computer Sciences, The University of Texas at Austin. September. `http://www.cs.utexas.edu/users/flame/`.

BILMES, J., ASANOVIC, K., CHIN, C., AND DEMMEL, J. 1997. Optimizing matrix multiply using PHiPAC: a portable, high-performace, ANSI C coding methodology. In *Proc. of the International Conference on SuperComputing, ACM SIGARC.*

CALVETTI, D. AND REICHEL, L. 1996. Application of ADI iterative methods to the restoration of noisy images. *SIAM J. Matrix Anal. Appl. 17*, 165–186.

DONGARRA, J., CROZ, J. D., DUFF, I., AND HAMMARLING, S. 1990. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft. 16*, 1–17.

DONGARRA, J. J., DU CROZ, J., HAMMARLING, S., AND DUFF, I. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16,* 1 (Mar.), 1–17.

DONGARRA, J. J., DU CROZ, J., HAMMARLING, S., AND HANSON, R. J. 1988. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw. 14,* 1 (Mar.), 1–17.

GARDINER, J., LAUB, A., AMATO, J., AND MOLER, C. 1992. Solution of the Sylvester matrix equation $AXB + CXD = E$. *ACM Trans. Math. Software 18*, 223–231.

GOLUB, G. AND VAN LOAN, C. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore.

GOLUB, G. H., NASH, S., AND VAN LOAN, C. F. 1979. A Hessenberg–Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control AC-24*, 909–913.

GUNNELS, J., HENRY, G., AND VAN DE GEIJN, R. 2001. A family of high-performance matrix multiplication algorithms. In *Computational Science - ICCS 2001, Part I, Lecture Notes in Computer Science 2073*, V. Alexander, J. Dongarra, B. Julianno, R. Renner, and C. Kenneth Tan, Eds. 51–60.

GUNNELS, J. A., GUSTAVSON, F. G., HENRY, G. M., AND VAN DE GEIJN, R. A. 2001. Flame: Formal linear algebra methods environment. *ACM Trans. Math. Softw. 27,* 4 (Dec.), 422–455.

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

HIGHAM, N. J. 2002. *Accuracy and Stability of Numerical Algorithms*, Second ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

JONSSON, I. AND KÅGSTRÖM, B. 2001. 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.

KÅGSTRÖM, B. AND POROMAA, P. 1992. Distributed and shared memory block algorithms for the triangular Sylvester equation with $\text{sep}^{-1}$ estimator. *SIAM J. Matrix Anal. Appl. 13,* 1, 90–101.

LAWSON, C. L., HANSON, R. J., KINCAID, D. R., AND KROGH, F. T. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw. 5,* 3 (Sept.), 308–323.

SIMA, V. 1996. *Algorithms for Linear-Quadratic Optimization.* Pure and Applied Mathematics, vol. 200. Marcel Dekker, Inc., New York, NY.

The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Mass, 01760 1993. *The MATLAB Control Toolbox, Version 3.0b.* The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Mass, 01760.

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

Received Month Year; revised Month Year; accepted Month Year