Automatic Derivation of Linear Algebra Algorithms with Application to Control Theory

Paolo Bientinesi[1], Sergey Kolos[2], and Robert van de Geijn[1]

[1] Department of Computer Sciences
The University of Texas at Austin
{pauldj, rvdg}@cs.utexas.edu
[2] The Institute for Computational Engineering and Sciences
The University of Texas at Austin
skolos@mail.utexas.edu

Abstract. It is our belief that the ultimate automatic system for deriving linear algebra libraries should be able to generate a set of algorithms starting from the mathematical specification of the target operation only. Indeed, one should be able to visit a website, fill in a form with information about the operation to be performed and about the target architectures, click the SUBMIT button, and receive an optimized library routine for that operation even if the operation has not previously been implemented. In this paper we show how close we are to what is probably regarded as an unreachable dream. We discuss the steps necessary to automatically obtain an algorithm starting from the mathematical abstract description of the operation to be solved. We illustrate how these steps have been incorporated into two prototype systems and we show the application of one the two systems to a problem from Control Theory: The Sylvester Equation. The output of this system is a description of an algorithm that can then be directly translated into code thanks to a set of API's we developed. The description can also be passed as input to a parallel performance analyzer to obtain optimized parallel routines [4].

1 Introduction

In a series of journal papers we have demostrated how formal derivation techniques can be applied to linear algebra to derive provably correct families of high performance algorithms [5, 8, 2]. In the paper *Rapid Development of High-Performance Linear Algebra Libraries*, also presented at this meeting, we illustrate the steps of a procedure which returns algorithms for linear algebra operations. It is beneficial for the reader to put these two documents side by side to better understand the following discussion.

While the procedure we described ensures the derived algorithm to be correct, it is the application of the procedure itself that is error prone. It involves tedious algebraic manipulations. As the complexity of the operation we want to implement increases, it becomes harder and harder to perform the procedure by hand.

We want to stress that the procedure does not introduce unnecessary elements of confusion. Operations once deemed "for experts only" can now be tackled by undergraduates, leaving more ambitious problems for the experts. Let us illustrate this concept with a concrete example: the solution to the Sylvester equation $AX + XB = C$.

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

Fig. 1. Worksheet for developing linear algebra algorithms.

Algorithms for solving the Sylvester equation have been known for about 30 years [1]. Nonetheless new variants are still discovered with some regularity and published in journal papers [3, 7]. The methodolody we describe in [2] has been applied to this operation yielding a family of 16 algorithms, including the algorithms already known as well as many undiscovered ones [8]. The only complication is that, as part of the derivation, complex matrix expression are introduced that require simplification, providing an opportunity for algebra mistakes to be made by a human. As a matter of fact we noted that one of the variants derived in [8], did not return the correct outcome when implemented and executed. Mistakes in simplifying expressions in order to determine the update were only detected when the updates were rederived with the aid of one of the automated systems described next.

2 Automatic Generation of Algorithms

In Figure 1 we show a generic "worksheet" for deriving linear algebra algorithms. The blanks in the worksheet are filled out by following a 8-step procedure, generating algorithms for linear algebra operations. A description of the 8 steps is given in the paper *Rapid Development of High-Performance Linear Algebra Libraries*, also presented at this conference. Here we present the steps again, looking at chances for automation.

Step 1: Determine P_{pre} and P_{post}. These two predicates are given as part of the specifications for the operation we want to implement; therefore they are the input to an automated system.

Step 2: Determine P_{inv}. Once the operands have been partitioned and substituted into the postcondition, with the aid of mathematical relations and simplifications we get the expression for the operation we want to implement as function of the exposed submatrices (partitions) of the input operands. We call this expression the *partitioned expression*.

Loop invariants are obtained by selecting a subset of the operations that appear in the partitioned expression.

Step 3: Determine loop-guard G. The loop invariant describes the contents of output variables at different points of the program. Upon completion of the loop, the loop invariant is true and the double lines (representing the boundaries of how far the computation has gotten) can be expected to reside at the edges of the partitioned operands. These two pieces of information, the loop invariant and the where boundaries are, can be exploited together to automatically deduce a loop-guard. If a loop guard cannot be found, the selected loop invariant is labelled as infeasible and no further steps are executed.

Step 4: Determine the initialization. We require the initialization not to involve any computation. It can be observed that, typically, by placing the double lines on the boundaries, the precondition implies the loop-invariant. This placement can be automated. If such a placement cannot be found, the selected loop invariant is labelled as infeasible and no further steps are executed.

Step 5: Determine how to move boundaries. How to traverse through the data is determined by relating the state of the partitioning after initialization to the loop-guard. Also, operands with a particular structure (triangular, symmetric matrices) can only be partitioned and traversed in a way that preserves the structure, thus limiting degrees of freedom. This determination can be automated.

Step 6: Determine P_{before}. This predicate is obtained by: 1) applying the substitution rules dictated by the **Repartion** statement to the loop invariant and 2) expanding and simplifying the expressions. Stage 1) is straightforward. Stage 2) requires symbolic computation tools. Mathematica [10] provides a powerful environment for the required algebraic manipulations, facilitating automation.

Step 7: Determine P_{after}. Computing this predicate is like the computation of the state P_{before} except with different substitution rules. In this case the rules are dictated by the **Continue ... with** statement. Therefore automation is possible.

Step 8: Determine the update S_U. The updates are determined by a comparison of the states P_{before} and P_{after}. Automation of this step is a process that involves a great deal of pattern matching, symbolic manipulations and requires a library of transformation rules for matrix expressions. While we believe that automation for this step is at least partially achievable, we feel that human intervention is desirable to supervise the process. For this step we envision an interactive system that suggests possible updates and drives user through the process.

3 Automated Systems

Initially, for a limited set of target operations, a fully automated system was prototyped. This system took a description of the partitioned expression as input and returned all the possible algorithms corresponding to the feasible loop invariants. This system provided the evidence that at least for simple operations all the steps in Section 1 can be automated. The biggest drawback of the system was that there was no easy way for a user to extend its functionality, to include other classes of input matrices, and to deal with more complex operations (involving the computation of the inverse, transpose, or the solution to linear systems).

In a second effort we developed a more interactive tool that guides the user through the derivation process. The input for this system is a loop invariant for the operation under consideration and the output is a partially filled worksheet (see Fig. 2. We designed this system aiming at modularity and generality. Strong evidence now exists that it can been used to semi-automatically generate all algorithms for all operations to which FLAME has been applied in the past. These include all the BLAS operations, all major factorization algorithms, matrix inversion, reductions to Hessemberg, tri and bi-diagonal forms, and a large number of operations that arise in control theory.

This semi-automated system plays an important role as part of a possible fully automated system: First, it automatically computes the predicates P_{before} and P_{after}. Second, it can express the status P_{after} as a function of P_{before}, thus pointing out to the user the updates to be computed. Finally, it can determine dependencies among the updates, thus avoiding reduntant computations and problems with data being overwritten.

Notice that once the operation we want to implement is expressed in terms of partitioned operands, it is feasible to list possible loop invariants. Not all the loop invariants are feasible though: not every loop invariant leads to an algorithm for the target operation. In order to decide whether a loop invariant is feasible, some computations needs to be done. This second system can be used as part of a more complete system to test loop invariants and determine whether they are feasible (thus producing an algorithm) or not (thus discarding it).

4 A Control Theory Example

We illustrate here how the semi-automated system can be used to derive, with little human intervention, a family of algorithms to solve the Triangular Sylvester Equation. The solution of such an equation is given by a matrix X that satisfies the equality $AX + XB = C$, where A and B are triangular matrices. We use the notation $X = \Gamma(A, B, C)$ to indicate that the matrix X is the solution of the Sylvester equation identified by the matrices A, B and C. Without loss of generality, in the following we assume that both matrices A and B are upper triangular.

As we mentioned in the previous section, the semi-automated system takes a loop invariant as input. Loop invariants are obtained from the partitioned expression of the operation we are solving. Partitioning the matrices A, B, C and X, respectively as $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right)$, $\left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right)$, $\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right)$ and $\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right)$, is then possible to state the partitioned expression for X, solution to the Triangular Sylvester Equation:

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

From this expression a list of loop invariants are systematically generated by selecting a subset of the operations to be performed. The simplest loop invariant is:

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

which identifies a computation of the solution matrix X in which the bottom left quadrant X_{BL} contains the solution of the equation $A_{BR}X_{BR} + X_{BR}B_{TL} = C_{BL}$; the algorithm proceeds expanding the quadrant X_{BL} in the top-right direction, until X_{BL} includes the whole matrix X and therefore contains the solution (when quadrant X_{BR} coincides with matrix X, matrices A_{BR}, B_{TL} and C_{BL} coincide with A, B and C).

In the remainder of the paper we concentrate on a slightly more complicated loop invariant:

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

which corresponds to an algorithm where the quadrant X_{BL} contains again the solution of the equation $A_{BR}X_{BR} + X_{BR}B_{TL} = C_{BL}$, and the quadrant X_{TL} is partially updated. Figure 2 shows the output algorithm generated by the automated system for this loop invariant.

The crucial expressions appear in the boxes labeled "Loop invariant before the updates" (LI-B4)

loop invariant before the updates:

$-\hat{A}_{02} \,.\, (\Omega(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20})) + \hat{C}_{00}$	$\hat{C}_{01}$	$\hat{C}_{02}$
$-\hat{A}_{12} \,.\, (\Omega(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20})) + \hat{C}_{10}$	$\hat{C}_{11}$	$\hat{C}_{12}$
$\Omega(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20})$	$\hat{C}_{21}$	$\hat{C}_{22}$

and "Loop invariant after the updates" (LI-Aft)

loop invariant after the update [Aft]

$-\hat{A}_{01} \,.\, \mathbf{AFT}_{1,0} + \mathbf{B4}_{0,0}$	$-\hat{A}_{01} \,.\, \mathbf{AFT}_{1,1} - \hat{A}_{02} \,.\, \mathbf{AFT}_{2,1} + \mathbf{B4}_{0,1}$	$\mathbf{B4}_{0,2}$
$\Omega(\hat{A}_{11}, \hat{B}_{00}, \mathbf{B4}_{1,0})$	$\Omega(\hat{A}_{11}, \hat{B}_{11}, (-\mathbf{AFT}_{1,0} \,.\, \hat{B}_{01} - \hat{A}_{12} \,.\, \mathbf{AFT}_{2,1} + \mathbf{B4}_{1,1}))$	$\mathbf{B4}_{1,2}$
$\mathbf{B4}_{2,0}$	$\Omega(\hat{A}_{22}, \hat{B}_{11}, (-\mathbf{AFT}_{2,0} \,.\, \hat{B}_{01} + \mathbf{B4}_{2,1}))$	$\mathbf{B4}_{2,2}$

These two predicates dictate the computations to be performed in the algorithm. LI-B4 expresses the current contents of the matrix X (i.e. the loop invariant), while LI-Aft indicates what X needs to contain -after- the updates to ensure that the selected

Operation: sylv3(A B C)

{Precondition: ...}

Partition

$$A \to \begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \qquad B \to \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \qquad C \to \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix}$$

where ...

loop invariant:

$$\begin{pmatrix} -\hat{A}_{TR} . \left(\Omega\left(\hat{A}_{BR}, \hat{B}_{TL}, \hat{C}_{BL}\right)\right) + \hat{C}_{TL} & \hat{C}_{TR} \\ \Omega\left(\hat{A}_{BR}, \hat{B}_{TL}, \hat{C}_{BL}\right) & \hat{C}_{BR} \end{pmatrix}$$

While ...

epartition

$$\begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \to \begin{pmatrix} \begin{pmatrix} \hat{A}_{00} & \hat{A}_{01} \\ 0 & \hat{A}_{11} \end{pmatrix} & \begin{pmatrix} \hat{A}_{02} \\ \hat{A}_{12} \end{pmatrix} \\ 0 & \hat{A}_{22} \end{pmatrix}, \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \to \begin{pmatrix} \hat{B}_{00} & \{\hat{B}_{01}, \hat{B}_{02}\} \\ 0 & \begin{pmatrix} \hat{B}_{11} & \hat{B}_{12} \\ 0 & \hat{B}_{22} \end{pmatrix} \end{pmatrix}, \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix} \to \left(\begin{pmatrix} \hat{C}_{00} \\ \hat{C}_{10} \\ \hat{C}_{20} \end{pmatrix} \right.$$

loop invariant before the updates:

$-\hat{A}_{02} . \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right) + \hat{C}_{00}$	$\hat{C}_{01}$	$\hat{C}_{02}$
$-\hat{A}_{12} . \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right) + \hat{C}_{10}$	$\hat{C}_{11}$	$\hat{C}_{12}$
$\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)$	$\hat{C}_{21}$	$\hat{C}_{22}$

UPDATES...

loop invariant after the update [Aft]

$-\hat{A}_{01} . \mathbf{AFT}_{1,0} + \mathbf{B4}_{0,0}$	$-\hat{A}_{01} . \mathbf{AFT}_{1,1} - \hat{A}_{02} . \mathbf{AFT}_{2,1} + \mathbf{B4}_{0,1}$	$\mathbf{B4}_{0,2}$
$\Omega\left(\hat{A}_{11}, \hat{B}_{00}, \mathbf{B4}_{1,0}\right)$	$\Omega\left(\hat{A}_{11}, \hat{B}_{11}, \left(-\mathbf{AFT}_{1,0} . \hat{B}_{01} - \hat{A}_{12} . \mathbf{AFT}_{2,1} + \mathbf{B4}_{1,1}\right)\right)$	$\mathbf{B4}_{1,2}$
$\mathbf{B4}_{2,0}$	$\Omega\left(\hat{A}_{22}, \hat{B}_{11}, \left(-\mathbf{AFT}_{2,0} . \hat{B}_{01} + \mathbf{B4}_{2,1}\right)\right)$	$\mathbf{B4}_{2,2}$

ontinue with

$$\begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \to \begin{pmatrix} \hat{A}_{00} & \{\hat{A}_{01}, \hat{A}_{02}\} \\ 0 & \begin{pmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & \hat{A}_{22} \end{pmatrix} \end{pmatrix}, \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \to \begin{pmatrix} \begin{pmatrix} \hat{B}_{00} & \hat{B}_{01} \\ 0 & \hat{B}_{11} \end{pmatrix} & \begin{pmatrix} \hat{B}_{02} \\ \hat{B}_{12} \end{pmatrix} \\ 0 & \hat{B}_{22} \end{pmatrix}, \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix} \to \left(\begin{matrix} \{\hat{C}_{00}, \hat{C}_{01}\} \\ \begin{pmatrix} \hat{C}_{10} & \hat{C}_{11} \\ \hat{C}_{20} & \hat{C}_{21} \end{pmatrix} \end{matrix} \right.$$

end while

Fig. 2. Algorithm returned by the semi-automatic system.

loop invariant holds at each iteration of the loop. For the sake of readability Figure 2 presents these predicates shortened, and the box "Updates" is left empty. Besides, LI-Aft presents a few visual cues to compress the otherwise long and unreadable expressions. We provide here a full description of both predicates and we explicitly state the updates which are encoded in LI-Aft.

The complete expression for the predicate LI-B4 is:

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

and we refer to the (i, j) quadrant of the right-hand side as $\mathrm{B4}_{ij}$. So for istance $\mathrm{B4}_{20}$ corresponds to the expression $\Gamma(A_{22}, B_{00}, C_{20})$.

The complete expression for the predicate LI-Aft is the daunting:

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

$$\left(\begin{array}{c|c||c} \begin{array}{l} C_{00} - A_{02}\Gamma(A_{22}, B_{00}, C_{20}) - \\ A_{01}\Gamma(A_{11}, B_{00}, \\ \quad C_{10} - A_{12}\Gamma(A_{22}, B_{00}, C_{20})) \end{array} & \begin{array}{l} C_{01} - \\ A_{01}\Gamma\big(A_{11}, B_{11}, C_{11} - \\ \quad \Gamma(A_{11}, B_{00}, C_{10} - A_{12}\Gamma(A_{22}, B_{00}, C_{20}))B_{01} - \\ \quad A_{12}\Gamma(A_{22}, B_{11}, C_{21} - \Gamma(A_{22}, B_{00}, C_{20})B_{01})\big) - \\ A_{02}\Gamma(A_{22}, B_{11}, C_{21} - \Gamma(A_{22}, B_{00}, C_{20})B_{01}) \end{array} & C_{02} \\ \hline\hline \begin{array}{l} \Gamma(A_{11}, B_{00}, \\ \quad C_{10} - A_{12}\Gamma(A_{22}, B_{00}, C_{20})) \end{array} & \begin{array}{l} \Gamma\big(A_{11}, B_{11}, C_{11} - \\ \quad \Gamma(A_{11}, B_{00}, C_{10} - A_{12}\Gamma(A_{22}, B_{00}, C_{20}))B_{01} - \\ \quad A_{12}\Gamma(A_{22}, B_{11}, C_{21} - \Gamma(A_{22}, B_{00}, C_{20})B_{01})\big) \end{array} & C_{12} \\ \hline \Gamma(A_{22}, B_{00}, C_{20}) & \Gamma(A_{22}, B_{11}, C_{21} - \Gamma(A_{22}, B_{00}, C_{20})B_{01}) & C_{22} \end{array}\right),$$

and the quadrants in the right-hand side of LI-Aft are identified by

$$\begin{array}{c|c|c} \mathrm{Aft}_{00} & \mathrm{Aft}_{01} & \mathrm{Aft}_{02} \\ \hline \mathrm{Aft}_{10} & \mathrm{Aft}_{11} & \mathrm{Aft}_{12} \\ \hline \mathrm{Aft}_{20} & \mathrm{Aft}_{21} & \mathrm{Aft}_{22} \end{array}.$$

Once the predicates LI-B4 and LI-Aft are known, we are one step away to have a complete algorithm. The updates S_U remain to be computed (Step 8 in Section 2). S_U are statements to be executed in a state in which the predicate LI-B4 holds and they have ensure that upon termination the predicate LI-Aft holds.

In this example, given the complexity of the expressions, such a task can be challenging even for experts, and is definitely prone to errors. A (semi-)automated system is useful, if not indispensable. Our system has a number of features to make Step 8 (discovering the updates S_U) as simple as possible:

- The expressions in LI-Aft are scanned to detect quantities contained in LI-B4. Such quantities are currently stored and therefore available to be used as operands; they are identified by a boxed grey highlight. For example, recognizing that the expression $\Gamma(A_{22}, B_{00}, C_{20})$ is contained in the quadrant $\mathrm{B4}_{20}$, the system would always display it as: $\boxed{\Gamma(A_{22}, B_{00}, C_{20})}$
- A quantity currently available (therefore higlighted) can be replaced by a label indicating the quadrant that contains it. This feature helps to shorten complicated

expressions. As an example, one instance of $\Gamma(A_{22}, B_{00}, C_{20})$ would be replaced by B4_{20}. Notice that $\Gamma(A_{22}, B_{00}, C_{20})$ appears in the quadrant Aft_{00}, as part of the expression $C_{00} - A_{02}\Gamma(A_{22}, B_{00}, C_{20})$; in this case the instance is not replaced by B4_{20} because the entire (and more complex) expression is recognized to appear in B4_{00}. Therefore, $C_{00} - A_{02}\Gamma(A_{22}, B_{00}, C_{20})$ is displayed as B4_{00}.

– Dependencies among quadrants are investigated. If the same computation appears in two or more quadrants, the system imposes an ordering to avoid redundant computations. Example: the quadrant Aft_{00}, after the replacements explained in the former two items, would look like $\text{B4}_{00} - A_{01}\Gamma\big(A_{11}, B_{00}, C_{10} - A_{12}\,\text{B4}_{20}\big)\big)$, and recognizing that the quantity $\Gamma\big(A_{11}, B_{00}, C_{10} - A_{12}\,\text{B4}_{20}\big)\big)$ is what the quadrant Aft_{10} has to contain at the end of the computation, the system leaves such an expression unchanged in quadrant Aft_{10} and instead replaces it in quadrant Aft_{00}, which would then be displayed as $\text{B4}_{00} - A_{01}\,\text{Aft}_{10}$.

The repeated application of all these steps yields a readable expression for LI-Aft, as shown in Figure 2. The updates are encoded in LI-Aft and can be made explicit by applying the following simple rules:

– The assignments are given by the componentwise assignment

$$\begin{array}{c|c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array} := \begin{array}{c|c|c} \text{Aft}_{00} & \text{Aft}_{01} & \text{Aft}_{02} \\ \hline \text{Aft}_{10} & \text{Aft}_{11} & \text{Aft}_{12} \\ \hline \text{Aft}_{20} & \text{Aft}_{21} & \text{Aft}_{22} \end{array}.$$

In our example it results:

$$\begin{array}{c|c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array} :=$$

$$\begin{array}{c|c|c} \text{B4}_{00} - A_{01}\,\text{Aft}_{10} & \text{B4}_{01} \text{ - } A_{01}\,\text{Aft}_{11} \text{ - } A_{02}\,\text{Aft}_{21} & \text{B4}_{02} \\ \hline \Omega(A_{11}, B_{00}, \text{B4}_{10}) & \Omega(A_{11}, B_{11}, \text{B4}_{11} - \text{Aft}_{10}\,B_{01} - A_{12}\,\text{Aft}_{21}) & \text{B4}_{12} \\ \hline \text{B4}_{20} & \Omega(A_{22}, B_{11}, \text{B4}_{21} - \text{Aft}_{20}\,B_{01}) & \text{B4}_{22} \end{array}$$

– Every assignment of the form $X_{ij} = \text{B4}_{ij}$ corresponds to a no-operation.

– Every assignment whose right-hand side presents one or more operands of the form Aft_{ij} has to be executed after the quadrant (i, j) has been computed. Once the expression in quadrant (i, j) has been computed, Aft_{ij} has to be rewritten as X_{ij}.

– Assignments with a right-hand side containing only non-highlighted expressions and/or operands of the form B4_{ij} can be computed immediately.

A valid set of updates for the current example is given by:

$$\begin{aligned} X_{10} &:= \Omega(A_{11}, B_{00}, \text{B4}_{10}) \\ X_{00} &:= \text{B4}_{00} - A_{01}X_{10} \\ X_{21} &:= \Omega(A_{22}, B_{11}, \text{B4}_{21} - X_{20}B_{01}) \\ X_{11} &:= \Omega(A_{11}, B_{11}, \text{B4}_{11} - X_{10}B_{01} - A_{12}X_{21}) \\ X_{01} &:= \text{B4}_{01} - A_{01}X_{11} - A_{02}X_{21}. \end{aligned}$$

The final algorithm is:

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

where A_{TL} is 0×0, B_{BR} is 0×0, C_{BL} is 0×0

while $\neg\text{SameSize}(C, C_{BL})$ **do**

Determine block size 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 $b_n \times b_n$, C_{11} is $b_m \times b_n$

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

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

5 Conclusion

In an effort to demonstrate that automatic derivation of linear algebra algorithm is achievable, we developed two (semi-)automated systems. The first system is fully automated but with very limited scope. It proved that at least for simple operations the FLAME methodology can be automated in all the steps. The second system is more interactive and we believe as general as the FLAME approach itself.

The described system has allowed us to automatically generate algorithms for most of the equations in a recent Ph.D. dissertation [6]. In that disseration, a number of important and challenging linear algebra problems arising in control theory are studies. For most of these problems, one algorithm and implementation is offered. By contrast, with the aid of our automated systems we were able to derive whole families of algorithms and their implementations (in Matlab Mscript as well as in C) collectively in a matter of hours. The implementations yielded correct answers for the first and all inputs with which they were tested. Moreover, parallel implementations can be just as easily created with the FLAME-like extension of our Parallel Linear Algebra Package (PLAPACK) [9]. We can thus claim that thanks to these tools the expertise of a Ph.D. candidate is no longer required for the development of parallel libraries for a broad class of linear algebra operations.

Additional information

For additional information on FLAME visit
`http://www.cs.utexas.edu/users/flame/`

Acknowledgments

This research is partially supported by NSF grants ACI-0305163 and CCF-0342369.

References

1. R. H. Bartels and G. W. Stewart. Solution of the matrix equation ax + xb = c. *Commun. ACM*, 15(9):820–826, 1972.
2. Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*. to appear.
3. Bo Kågström and Peter Poromaa. Lapack-style algorithms and software for solving the generalized sylvester equation and estimating the separation between regular matrix pairs. *ACM Transactions on Mathematical Software*, 22(1):78–103, 1996.
4. John Gunnels. *A Systematic Approach to the Design and Analysis of Parallel Dense Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, UT-Austin, in preparation.
5. John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.
6. Isak Jonsson. *Recursive Blocked Algorithms, Data Structures, and High-Performance Software for Solving Linear Systems and Matrix Equations*. PhD thesis, Dept. Computing Science, Umeå University, SE-901 87, Sweden., 2003.
7. Isak Jonsson and Bo Kågström. Recursive blocked algorithms for solving triangular systems—part i: one-sided and coupled sylvester-type matrix equations. *ACM Transactions on Mathematical Software*, 28(4):392–415, 2002.
8. Enrique S. Quintana-Ortí and Robert A. van de Geijn. Formal derivation of algorithms: The triangular Sylvester equation. *ACM Transactions on Mathematical Software*, 29(2):218–243, June 2003.
9. Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.
10. Stephen Wolfram. *The Mathematica Book: 3rd Edition*. Cambridge University Press, 1996.