Deriving Dense Linear Algebra Libraries

Paolo Bientinesi[1], John Gunnels[2], Margaret Myers[3], Enrique Quintana-Ortí[4],
Tyler Rhodes[3], Robert van de Geijn[3], and Field G. Van Zee[3]

[1]RWTH Aachen University, Aachen, Germany,

[2]IBM T.J. Watson Research Center, Yorktown Heights, NY,

[3]The University of Texas at Austin, Austin, TX,

[4] Universidad Jaime I, Castellón, Spain

Abstract. Starting in the late 1960s computer scientists including Dijkstra and Hoare advocated goal-oriented programming and the formal derivation of algorithms. The chief impediment to realizing this for loop-based programs was that *a priori* determination of loop-invariants, a prerequisite for developing loops, was a task too complex for any but the simplest of operations. Around 2000, these techniques were for the first time successfully applied to the domain of high-performance dense linear algebra libraries. This has led to a multitude of papers, mostly published in the ACM Transactions for Mathematical Software, a system for the mechanical derivation of algorithms, and a high-performance linear algebra library, `libflame`, that includes more than a thousand variants of algorithms for more than a hundred linear algebra operations. To our knowledge, this success story has unfolded with limited awareness on the part the formal methods community. This paper reports on ten years of experience and is meant to raise that awareness.

Keywords: Formal derivation; Linear algebra libraries; Scientific computing

1. Introduction

Linear algebra libraries reside at the bottom of the scientific computing food chain. While most practical applications give rise to sparse linear algebra problems, a significant number of them spends most computational time solving dense matrix problems. Even sparse linear algebra problems often have dense subproblems to be solved. As a result, LAPACK [ABB+99], a package for dense matrix operations, developed in the late 1980s and early 1990s, is undoubtedly the most commonly used library in this field.

Since 2000, the FLAME project at The University of Texas at Austin, Universidad Jaume I (Spain), and RWTH Aachen University (Germany) has been pursuing a replacement of LAPACK, `libflame` [Zee09]. The domain poses a few interesting challenges: scientific computing tends to exploit the latest architectures,

demanding the highest possible performance. This requires the design of loop-based algorithms that cast most computation in terms of high-performing matrix-matrix operations (like matrix-matrix multiplication). The loop steps through matrices with block sizes chosen so as to optimize the reuse of data in caches. For a specific operation, there are often multiple algorithmic variants, with one algorithmic variant matching a given architecture better than the others, yielding higher performance. Thus, a library should incorporate multiple loop-based algorithms for a given operation so that the best one can be chosen. The LAPACK library does not include such a wide variety of algorithms. This brings up the question of how to systematically find algorithmic variants for a given operation. The formal derivation of loop-based algorithms turns out to be the answer [Bie06, BGM+05, BGdG, Gun01, GGHvdG01, GvdG01, QOvdG03, vdGQO08], as we will illustrate.

This paper does not provide a scholarly treatment of the field of derivation of algorithms. All we have ever needed to develop the described techniques is given in the text by Gries [Gri81], which itself is based on the works of Dijkstra [Dij68, Dij76] and Hoare [Hoa69]. What the paper does provide is what we believe to be an excellent practical example of the application of formal derivation of loops to the domain of dense linear algebra.

2. Derivation of Linear Algebra Algorithms

In this section, we walk the reader through the derivation process. The procedure is completely routine for us, as we have applied it to more than a hundred operations, yielding more than a thousand routines that are part of the `libflame` library. We use the solution of the triangular Lyapunov equation as our motivating example. A reader who is not well-versed in linear algebra should not worry: the methology is systematic to the point where one does not need to be an expert in order to apply it. A more basic treatment that targets novices and has been used at the undergraduate level can be found in [vdGQO08].

2.1. The FLAME methodology

A fundamental insight in our project was the realization that the Fundamental Invariance Theorem [Gri81], used to prove the correctness of a loop in a program, can be formulated as a *worksheet* that is systematically filled out, first with assertions (predicates) and then with commands (imperative statements) [BGM+05]. The worksheet, given in Figure 1, will be filled out with predicates that indicate prescribed states and commands that achieve those states. It is filled out in the order indicated in the column marked by **Step**. The predicates P_{pre}, P_{post}, P_{inv}, and G represent the precondition, postcondition, loop-invarant, and loop-guard, respectively. The loop-invariant has to be *true* in four different places: before and after the loop, and at the top and bottom of the loop body. The other parts of the worksheet will become obvious as we fill it out for our example.

2.2. Example: the solution of the triangular Lyapunov equation

We now show how the methodology is applied to a prototypical example: the solution of the triangular Lyapunov equation given by $U^T X + XU + C = 0$ (or, alternatively, $U^T X + XU = -C$), where U is an upper triangular matrix and C and X are symmetric matrices. Here the superscript "T" indicates matrix transposition. The solution X is to be computed. Because of symmetry, only the upper triangular part of C is stored and is overwritten with the upper triangular part of X. This operation is preceded by a pre-processing phase (not discussed in this paper) that transforms the general (non-triangular) time-continuous Lyapunov equation (an operation encountered in control theory [Kha02]) to the given triangular Lyapunov form.

2.3. Filling out the worksheet

At this point, the reader should imagine the worksheet in Figure 2 as being empty and the steps detailed below as filling out the worksheet in the indicated order.

Step 1: The precondition and postcondition.

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

Fig. 1. Blank FLAME worksheet used to derive algorithms.

The precondition is given by $C = \hat{C}$ while the postcondition is $C = X \wedge U^T X + X U = -\hat{C}$. Here the $\hat{}$ is needed to be able to reason about the current contents (state) of variable C relative to the initial contents, $\hat{C}$. For brevity, properties of the matrices (like triangular structure and sizes) are not expressed in the precondition nor in other predicates.

Step 2: Deriving the loop-invariants. A fundamental insight is that many algorithms sweep through matrices (arrays) in a systematic and predictable fashion. In this example, all matrices are either triangular or symmetric and are partitioned into quadrants since this exposes regions of the matrices that are either (implicitly of explicitly) zero, for the triangular matrix, or not stored, for the symmetric matrices:

$$U = \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right), X = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right), C = \left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right), \text{ and } \hat{C} = \left(\begin{array}{c|c} \hat{C}_{TL} & \hat{C}_{TR} \\ \hline \star & \hat{C}_{BR} \end{array} \right), (1)$$

where U_{TL}, X_{TL}, C_{TL}, and $\hat{C}_{TL}$ are all conformal (of the same size) and square. Here TL, TR, BL, and BR stand for "top-left", "top-right", "bottom-left", and "bottom-right", respectively. The 0 and $\star$ indicate submatrices that are entirely zero or not stored, respectively. As the algorithm progresses, the top-left (TL) quadrants will grow from empty (0×0) to encompassing the entire matrix.

Substituting the partitioned matrices in (1) into the postcondition yields the expression in Figure 3, which we call the *Partitioned Matrix Expression* [Bie06, vdGQO08] (PME). It is a recursive definition of the operation in terms of the exposed submatrices.

The PME expresses the computation to be performed (which must make the postcondition *true*) in terms of the quadrants. Observe that as long as the loop has not finished, only *part* of the computation expressed by the PME is satisfied: to come up with potential loop-invariants, one deletes some of the subexpressions in the PME, as illustrated in Figure 4. As long as the expression that is left is a valid expression that has the correct size, it is a candidate. Some potential loop-invariants are such that subsequent steps cannot be performed, which means that they do not yield an (admissible) algorithm. For example, if only the original contents of the matrix are left after deleting subexpressions from the PME, the loop can clearly not complete

Step	Annotated Algorithm: $C := \text{LYAP\_UNB}(U, C)$
1a	$\left\{ C = \hat{C} \right\}$
4	**Partition** $U \to \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right)$, $X \to \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right)$, $C \to \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right)$ **where** U_{TL} is 0×0, X_{TL} is 0×0, C_{TL} is 0×0, $\hat{C}_{TL}$ is 0×0
2	$\left\{ \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{cases} \right\}$
3	**while** $m(U_{TL}) < m(U)$ **do**
2,3	$\left\{ \left(\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{cases} \right) \wedge (m(U_{TL}) < m(U)) \right\}$
5a	**Repartition** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) \to \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array} \right)$, $\left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \to \left(\begin{array}{c\|c\|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array} \right)$, $\cdots$ **where** $v_{11}, \chi_{11}, \gamma_{11}$ are scalars
6	$\left\{ \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline \star & \gamma_{11} & c_{12}^T \\ \hline \star & \star & C_{22} \end{array} \right) = \left(\begin{array}{c\|c\|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array} \right) \wedge \begin{cases} U_{00}^T X_{00} + X_{00} U_{00} = -\hat{C}_{00} \\ U_{00}^T x_{01} + X_{00} u_{01} + x_{01} v_{11} = -\hat{c}_{01} \\ U_{00}^T X_{02} + X_{00} U_{02} + x_{01} u_{12}^T + X_{02} U_{22} = -\hat{C}_{02} \\ \chi_{11} = \hat{\gamma}_{11} \wedge x_{12}^T = \hat{c}_{12}^T \wedge X_{22} = \hat{C}_{22} \end{cases} \right\}$
8	$\gamma_{11} := (-\gamma_{11} - 2u_{01}^T c_{01})/(2v_{11})$ $c_{12}^T := -c_{12}^T - u_{01}^T C_{02} - c_{01}^T U_{02} - \gamma_{11} u_{12}^T$ Solve $v_{11} x_{12}^T + x_{12}^T U_{22} = c_{12}^T$ overwriting c_{12}^T with x_{12}^T
5b	**Continue with** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array} \right)$, $\left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c\|c\|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array} \right)$, $\cdots$
7	$\left\{ \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline \star & \gamma_{11} & c_{12}^T \\ \hline \star & \star & C_{22} \end{array} \right) = \left(\begin{array}{c\|c\|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array} \right) \right. \\ \left. \wedge \begin{cases} U_{00}^T X_{00} + X_{00} U_{00} = -\hat{C}_{00} \\ U_{00}^T x_{01} + X_{00} u_{01} + x_{01} v_{11} = -\hat{c}_{01} \\ U_{00}^T X_{02} + X_{00} U_{02} + x_{01} u_{12}^T + X_{02} U_{22} = -\hat{C}_{02} \\ 2u_{01}^T x_{01} + 2v_{11} \chi_{11} = -\hat{\gamma}_{11} \\ u_{01}^T X_{02} + v_{11} x_{12}^T + x_{01}^T U_{02} + \chi_{11} u_{12}^T + x_{12}^T U_{22} = -\hat{c}_{12}^T \\ X_{22} = \hat{C}_{22} \end{cases} \right\}$
2	$\left\{ \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{cases} \right\}$
	endwhile
2,3	$\left\{ \left(\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array} \right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array} \right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{cases} \right) \wedge \neg(m(U_{TL}) < m(U)) \right\}$
1b	$\left\{ U^T X + XU = -C \right\}$

Fig. 2. Worksheet for deriving the unblocked algorithm for solving the triangular Lyapunov equation corresponding to Loop-invariant 3.

Substituting the partitioned operands in (1) into $U^T X + XU = -\hat{C}$ (the postcondition), yields

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right)$$

$$\wedge \left(\begin{array}{c|c} U_{TL}^T & 0 \\ \hline U_{TR}^T & U_{BR}^T \end{array}\right)\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) + \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right)\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) = \left(\begin{array}{c|c} -\hat{C}_{TL} & -\hat{C}_{TR} \\ \hline \star & -\hat{C}_{BR} \end{array}\right)$$

which (by linear algebra manipulation) is equivalent to

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ U_{BR}^T X_{BR} + X_{BR} U_{BR} = -\hat{C}_{BR} - (U_{TR}^T X_{TR} + X_{TR}^T U_{TR}) \end{array} \right.$$

Fig. 3. The Partitioned Matrix Expression (PME) (recursive definition of the operation) for overwriting C with the solution of the triangular Lyapunov equation.

Loop-invariant 1:

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ \cancel{U_{TL}^T X_{TR} + X_{TR} U_{BR}} = +\hat{C}_{TR} \cancel{- X_{TL} U_{TR}} \\ \cancel{U_{BR}^T X_{BR} + X_{BR} U_{BR}} = +\hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{array} \right.$$

Loop-invariant 2:

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ \cancel{U_{TL}^T X_{TR} + X_{TR} U_{BR}} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T X_{BR} + X_{BR} U_{BR}} = +\hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{array} \right.$$

Loop-invariant 3:

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T X_{BR} + X_{BR} U_{BR}} = +\hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{array} \right.$$

Loop-invariant 4:

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T X_{BR} + X_{BR} U_{BR}} = -\hat{C}_{BR} - (U_{TR}^T X_{TR} + X_{TR}^T U_{TR}) \end{array} \right.$$

Fig. 4. Loop-invariants for the triangular Lyapunov equation.

---

in a state where the postcondition holds; this exhibits itself when no loop-guard can be found in Step 3. Invariants are thus systematically derived from the PME.

We will now focus on one loop-invariant (Loop-invariant 3) as we fill out the remainder of the worksheet in Figure 2. The methodology yields algorithms corresponding to the other loop-invariants in an analogous fashion.

Step 3: Loop guard G. We know that after the loop completes, $\{P_{inv} \wedge \neg G\}$ is *true*. No commands exists between this and the postcondition $\{P_{post}\}$. Thus, G must be chosen so that

$$\left(\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left(\left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{array} \right.\right) \wedge \neg G \right)$$

implies $U^T X + XU = -C$. This yields the (nonunique) choice for the loop-guard: $G = (m(U_{TL}) < m(U))$,

where $m(\cdot)$ returns the row dimension of the argument. (An implicit assumption here is that matrices U_{TL}, X_{TL}, C_{TL}, and $\hat{C}_{TL}$ are always kept conformal, i.e., of the same size and square.)

Step 4: Initialization. The initialization is an indexing step: the matrices are partitioned as in (1). The fact that this must place the variables in a state where P_{inv} holds dictates the choice wherein the top-left quadrants are 0×0 (empty). There could be other choices for the initialization, but those would invariably involve performing computation with the matrices, altering their contents.

Step 5: Moving through the matrices. In Steps 5a and 5b, submatrices are exposed so that we make progress through the matrices. Here, thick lines have semantic meaning: a new row and column are exposed. Updates will happen in the loop body, and then that row and column are moved across the thick line to capture the traversal through the matrices.

In exposing submatrices, we use notational conventions that allow special properties of the submatrices to be easily recognized: lower case Greek letters denote scalars, lower case Roman letters denote column vectors, and upper case Roman letter denote matrices. Submatrices like u_{12}^T can be easily recognized as being part of a row and hence a row vector (transposed column vector).

The fact that the top-left quadrant is initially empty (0×0) and must eventually envelop the entire matrix dictates how the algorithm traverses through the matrices. The traversal through the matrix, together with the finite size of the operands, means that there is a natural, monotonically decreasing loop-bound function, $t = n - m(U_{TL})$, that is bounded below. This ensures that the loop terminates.

Step 6: State before the update. The commands in Step 5a are merely indexing operations. Since no computation occurs between the top of the loop and Step 6 in the worksheet, the state of the submatrices that are exposed by Step 5a can be determined by textual substitution and linear algebra manipulation, as illustrated in Figure 5. This yields the state described by Step 6. The invariant together with the repartitioning in Step 5a dictates the predicate in Step 6.

Step 7: State after the update. Similarly, the commands in Step 5b are merely indexing operations. Since the invariant must again hold, the state in Step 7 can be systematically derived by textual substitution of the submatrices in Step 5b into the loop-invariant and linear algebra manipulation, as illustrated in Figure 6. The loop-invariant together with the redefinition of the quadrants in Step 5b dictate the predicate in Step 7.

Step 8: Update. The update in Step 8 is now dictated by the state that the variables are in at Step 6 and the state that they must be in at Step 7:

- C_{00} already contains X_{00}, the solution to $U_{00}^T X_{00} + X_{00} U_{00} = -\hat{C}_{00}$, and hence is not updated.

- c_{01} already contains x_{01}, the solution to $U_{00}^T x_{01} + X_{00} u_{01} + x_{01} v_{11} = -\hat{c}_{01}$, and hence is not updated.

- C_{02} already contains X_{02}, the solution to $U_{00}^T X_{02} + X_{00} U_{02} + x_{01} u_{12}^T + X_{02} U_{22} = -\hat{C}_{02}$, and hence is not updated.

- γ_{11} contains $\chi_{11} = \hat{\gamma}_{11}$ and needs to be overwritten by the solution, χ_{11}, of $2 u_{01}^T x_{01} + 2 v_{11} \chi_{11} = -\hat{\gamma}_{11}$. Recognizing that at this point c_{01} contains x_{01} and γ_{11} contains $\hat{\gamma}_{11}$, this can be accomplished by updating γ_{11} with $\gamma_{11} := (-\gamma_{11} - 2 u_{01}^T c_{01})/(2 v_{11})$, where := is used for assignment.

- c_{12}^T holds $x_{12}^T = \hat{c}_{12}^T$ and needs to be overwritten with the solution, x_{12}^T, of $u_{01}^T X_{02} + v_{11} x_{12}^T + x_{01}^T U_{02} + \chi_{11} u_{12}^T + x_{12}^T U_{22} = -\hat{c}_{12}^T$. Recognizing that X_{02} has overwritten C_{02}, etc., this can be accomplished by updating it with the solution, x_{12}^T, of $v_{11} x_{12}^T + x_{12}^T U_{22} = -c_{12}^T - u_{01}^T C_{02} - c_{01}^T U_{02} - \gamma_{11} u_{12}^T$:

  - First, we update $c_{12}^T := -c_{12}^T - u_{01}^T C_{02} - c_{01}^T U_{02} - \gamma_{11} u_{12}^T$.
  - Next, we compute $c_{12}^T := c_{12}^T (v_{11} I + U_{22})^{-1}$. This requires the solution of a triangular system of equations, since it is equivalent to computing the solution to $(v_{11} I + U_{22})^T x_{12} = c_{12}$ and overwriting c_{12}^T with x_{12}^T.

We reiterate that the state in Step 6 and the desired state in Step 7 dictate how the variables (submatrices of C) must be updated.

Resulting algorithm. The resulting algorithm, stripped of the annotations that were used to derive it, is given in Figure 7 (left), executing only the commands indicated under Variant 3.

Step 5a, given by

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \quad \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} C_{00} & c_{01} & C_{02} \\ \hline \star & \gamma_{11} & c_{12}^T \\ \hline \star & \star & C_{22} \end{array}\right), \quad \left(\begin{array}{c|c} \hat{C}_{TL} & \hat{C}_{TR} \\ \hline \star & \hat{C}_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} \hat{C}_{00} & \hat{c}_{01} & \hat{C}_{02} \\ \hline \star & \hat{\gamma}_{11} & \hat{c}_{12}^T \\ \hline \star & \star & \hat{C}_{22} \end{array}\right),$$

expresses

$$\left(\begin{array}{c|c} U_{TL} \rightarrow U_{00} & U_{TR} \rightarrow \left(\begin{array}{c|c} u_{01} & U_{02} \end{array}\right) \\ \hline 0 & U_{BR} \rightarrow \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right) \end{array}\right), \quad \left(\begin{array}{c|c} X_{TL} \rightarrow X_{00} & X_{TR} \rightarrow \left(\begin{array}{c|c} x_{01} & X_{02} \end{array}\right) \\ \hline \star & X_{BR} \rightarrow \left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline \star & X_{22} \end{array}\right) \end{array}\right),$$

$$\left(\begin{array}{c|c} C_{TL} \rightarrow C_{00} & C_{TR} \rightarrow \left(\begin{array}{c|c} c_{01} & C_{02} \end{array}\right) \\ \hline \star & C_{BR} \rightarrow \left(\begin{array}{c|c} \gamma_{11} & c_{12}^T \\ \hline \star & C_{22} \end{array}\right) \end{array}\right), \quad \left(\begin{array}{c|c} \hat{C}_{TL} \rightarrow \hat{C}_{00} & \hat{C}_{TR} \rightarrow \left(\begin{array}{c|c} \hat{c}_{01} & \hat{C}_{02} \end{array}\right) \\ \hline \star & \hat{C}_{BR} \rightarrow \left(\begin{array}{c|c} \hat{\gamma}_{11} & \hat{c}_{12}^T \\ \hline \star & \hat{C}_{22} \end{array}\right) \end{array}\right).$$

Substituting these into the state of the variables at the top of the loop (the invariant) given by

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \left\{ \begin{array}{l} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{array} \right.$$

yields the state of the exposed submatrices:

$$\left(\begin{array}{c|c} C_{00} & \left(\begin{array}{c|c} c_{01} & C_{02} \end{array}\right) \\ \hline \left(\begin{array}{c} \star \\ \hline \star \end{array}\right) & \left(\begin{array}{c|c} \gamma_{11} & c_{12}^T \\ \hline \star & C_{22} \end{array}\right) \end{array}\right) = \left(\begin{array}{c|c} X_{00} & \left(\begin{array}{c|c} x_{01} & X_{02} \end{array}\right) \\ \hline \left(\begin{array}{c} \star \\ \hline \star \end{array}\right) & \left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline \star & X_{22} \end{array}\right) \end{array}\right)$$

$$\wedge \left\{ \begin{array}{l} U_{00}^T X_{00} + X_{00} U_{00} = -\hat{C}_{00} \\[1em] U_{00}^T \left(\begin{array}{c|c} x_{01} & X_{02} \end{array}\right) + \left(\begin{array}{c|c} x_{01} & X_{02} \end{array}\right) \left(\begin{array}{c|c} v_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right) = -\left(\begin{array}{c|c} \hat{c}_{01} & \hat{C}_{02} \end{array}\right) - X_{00} \left(\begin{array}{c|c} u_{01} & U_{02} \end{array}\right) \\[1em] \left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline \star & X_{22} \end{array}\right) = \left(\begin{array}{c|c} \hat{\gamma}_{11} & \hat{c}_{12}^T \\ \hline \star & \hat{C}_{22} \end{array}\right). \end{array} \right.$$

Algebraic manipulation of the above expression yields Step 6 in the worksheet.

Fig. 5. Systematic derivation of the state of the variables at Step 6 in Figure 2.

2.4. Other algorithms

Other algorithms are derived from the other loop-invariants. In addition, blocked algorithms, which cast most computation in terms of matrix-matrix operations and hence can attain higher performance, can be derived by moving through the matrix several rows and columns at a time. Resulting algorithms are given in Figure 7. In the blocked algorithms, the operation

$$\text{Solve } U_{II}^T X_{IJ} + X_{IJ} U_{JJ} = C_{IJ}$$

requires the solution of the Sylvester equation. Algorithms for that operation were derived in [QOvdG03].

Substituting the redefinition of quadrants in Step 5b

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right), \quad \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} C_{00} & c_{01} & C_{02} \\ \hline \star & \gamma_{11} & c_{12}^T \\ \hline \star & \star & C_{22} \end{array}\right), \quad \left(\begin{array}{c|c} \hat{C}_{TL} & \hat{C}_{TR} \\ \hline \star & \hat{C}_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} \hat{C}_{00} & \hat{c}_{01} & \hat{C}_{02} \\ \hline \star & \hat{\gamma}_{11} & \hat{c}_{12}^T \\ \hline \star & \star & \hat{C}_{22} \end{array}\right)$$

into the desired state of the variables at the bottom of the loop (the invariant)

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ X_{BR} = \hat{C}_{BR} \end{cases}$$

results in

$$\left(\begin{array}{c|c|c} C_{00} & c_{01} & C_{02} \\ \hline \star & \gamma_{11} & c_{12}^T \\ \hline \star & \star & C_{22} \end{array}\right) = \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline \star & \chi_{11} & x_{12}^T \\ \hline \star & \star & X_{22} \end{array}\right)$$

$$\wedge \begin{cases} \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array}\right)^T \left(\begin{array}{c|c} X_{00} & x_{01} \\ \hline \star & \chi_{11} \end{array}\right) + \left(\begin{array}{c|c} X_{00} & x_{01} \\ \hline \star & \chi_{11} \end{array}\right) \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array}\right) = -\left(\begin{array}{c|c} \hat{C}_{00} & \hat{c}_{01} \\ \hline \star & \hat{\gamma}_{11} \end{array}\right) \\ \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & v_{11} \end{array}\right)^T \left(\begin{array}{c} X_{02} \\ \hline x_{12}^T \end{array}\right) + \left(\begin{array}{c} X_{02} \\ \hline x_{12}^T \end{array}\right) U_{22} = -\left(\begin{array}{c} \hat{C}_{02} \\ \hline \hat{c}_{12}^T \end{array}\right) - \left(\begin{array}{c|c} X_{00} & x_{01} \\ \hline \star & \chi_{11} \end{array}\right) \left(\begin{array}{c} U_{02} \\ \hline u_{12}^T \end{array}\right) \\ X_{22} = \hat{C}_{22}. \end{cases}$$

Algebraic manipulation yields the expression in Step 7.

Fig. 6. Systematic derivation of the state of the variables at Step 7 in Figure 2.

2.5. Discussion

It is the notation we use that facilitates the derivation process: by presenting submatrices rather than index ranges, the derived algorithm avoids much of the indexing clutter (both in the presentation of the algorithm and the details of the proof of correctness) that is typically found in conventional loop-based algorithms. Indeed, the algorithm exposes only one loop even though it requires approximately n^3 floating point operations. The other loops are hidden inside of the linear algebra operations that form the body of the loop. Algorithms for these operations themselves can be, and have been, formally derived, using the same techniques as described in this paper.

2.6. From algorithm to code

Using a correct algorithm as the basis for your implementation does not guarantee that the resulting code will be correct. To preserve the correctness of the algorithm as we translate it to code, we defined APIs for different languages such that the code closely resembles the algorithm [BQOvdG05, vdGQO08, VZCvdG+09]. An example of this is given in Figure 8. In that figure, unblocked Variant 3 is coded in M-script, the scripting language of Matlab [MLB87].

We created a handful of routines that partition and repartition the matrices. White-space is used to make the code resemble the algorithm as closely as possible. The code in Figure 8 gave the correct answer the first time it was run. We have developed APIs for the C programming language [BQOvdG05, vdGQO08,

Algorithm: $C := \text{LYAP\_UNB}(U, C)$	**Algorithm:** $C := \text{LYAP\_BLK}(U, C)$
Partition $U \to \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $X \to \cdots$ where U_{TL} is 0×0, X_{TL} is 0×0, C_{TL} is 0×0 **while** $m(U_{TL}) < m(U)$ **do** **Repartition** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \to \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$ where $v_{11}, \chi_{11}, \gamma_{11}$ are scalars	**Partition** $U \to \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $X \to \cdots$ where U_{TL} is 0×0, X_{TL} is 0×0, C_{TL} is 0×0 **while** $m(U_{TL}) < m(U)$ **do** **Determine block size** b **Repartition** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \to \left(\begin{array}{c\|c\|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$ where U_{11}, X_{11}, C_{11} are $b \times b$
Variant 1 $c_{01} := -c_{01} - C_{00}u_{01}$ Solve $U_{00}^T c_{01} + c_{01}v_{11} = c_{01}$ $\gamma_{11} := -\gamma_{11} - u_{01}^T c_{01} - c_{01}^T u_{01}$ $\gamma_{11} := \gamma_{11}/(2v_{11})$ **Variant 2** Solve $U_{00}^T c_{01} + c_{01}v_{11} = c_{01}$ $\gamma_{11} := -\gamma_{11} - u_{01}^T c_{01} - c_{01}^T u_{01}$ $\gamma_{11} := \gamma_{11}/(2v_{11})$ $C_{02} := C_{02} - c_{01}u_{12}^T$ $c_{12}^T := -c_{12}^T - \gamma_{11}u_{12}^T - c_{01}^T U_{02}$ **Variant 3** $\gamma_{11} := -\gamma_{11} - 2u_{01}^T c_{01}$ $\gamma_{11} := \gamma_{11}/(2v_{11})$ $c_{12}^T := -c_{12}^T - u_{01}^T C_{02} - c_{01}^T U_{02} - \gamma_{11}u_{12}^T$ Solve $v_{11}c_{12}^T + c_{12}^T U_{22} = c_{12}^T$ **Variant 4** $\gamma_{11} := \gamma_{11}/(2v_{11})$ $c_{12}^T := c_{12}^T - \gamma_{11}u_{12}^T$ Solve $v_{11}c_{12}^T + c_{12}^T U_{22} = c_{12}^T$ $C_{22} := C_{22} - u_{12}c_{12}^T - c_{12}u_{12}^T$	**Variant 1** $C_{01} := -C_{01} - C_{00}U_{01}$ Solve $U_{00}^T X_{01} + X_{01}U_{11} = C_{01}$ $C_{11} := -C_{11} - U_{01}^T C_{01} - C_{01}^T U_{01}$ Solve $U_{11}^T X_{11} + X_{11}U_{11} = C_{11}$ **Variant 2** Solve $U_{00}^T X_{01} + X_{01}U_{11} = C_{01}$ $C_{11} := -C_{11} - U_{01}^T C_{01} - C_{01}^T U_{01}$ Solve $U_{11}^T X_{11} + X_{11}U_{11} = C_{11}$ $C_{02} := C_{02} - C_{01}U_{12}$ $C_{12} := -C_{12} - C_{11}U_{12} - C_{01}^T U_{02}$ **Variant 3** $C11 := -C_{11} - U_{01}^T C_{01} - C_{01}^T U_{01}$ Solve $U_{11}^T X_{11} + X_{11}U_{11} = C_{11}$ $C_{12} := -C_{12} - U_{01}^T C_{02} - C_{01}^T U_{02} - X_{11}U_{12}$ Solve $U_{11}^T X_{12} + X_{12}U_{22} = C_{12}$ **Variant 4** Solve $U_{11}^T X_{11} + X_{11}U_{11} = C_{11}$ $C_{12} := C_{12} - C_{11}U_{12}$ Solve $U_{11}^T X_{12} + X_{12}U_{22} = C_{12}$ $C_{22} := C_{22} - U_{12}^T C_{12} - C_{12}^T U_{12}$
Continue with $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & v_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$ **endwhile**	**Continue with** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$ **endwhile**

Fig. 7. Algorithms for computing the solution to the triangular Lypunov equation. Left: unblocked algorithms. Right: blocked algorithms.

VZCvdG$^+$09, Zee09]. Additionally, the recently developed Elemental library for distributed memory architectures uses a similar API [PvdGB].

3. Performance

In Figure 9, we show the performance of different algorithmic variants that are implemented as part of `libflame`, blocked and unblocked, on a Dell PowerEdge R900 server, using four cores and double precision arithmetic. We report performance in GFLOPS (billions of floating point operations per second) by taking the known floating point operation count for this operation, n^3, dividing it by the time required to complete

```
function [ C_out ] = lyap_unb_var3( U, C )
  [ UTL, UTR, ...
    UBL, UBR ] = FLA_Part_2x2( U, ...
                                 0, 0, 'FLA_TL' );
  [ CTL, CTR, ...
    CBL, CBR ] = FLA_Part_2x2( C, ...
                                 0, 0, 'FLA_TL' );
  while ( size( CTL, 1 ) < size( C, 1 ) )
    [ U00, u01,        U02,  ...
      u10t, upsilon11, u12t, ...
      U20,  u21,       U22 ] = ...
          FLA_Repart_2x2_to_3x3( UTL, UTR, ...
                                 UBL, UBR, ...
                                 1, 1, 'FLA_BR' );
    [ C00,  c01,       C02,  ...
      c10t, gamma11, c12t, ...
      C20,  c21,       C22 ] = ...
          FLA_Repart_2x2_to_3x3( CTL, CTR, ...
                                 CBL, CBR, ...
                                 1, 1, 'FLA_BR' );
    %--------------------------------------------%
    gamma11 = -gamma11 - 2 * u01' * c01;
    gamma11 = gamma11 / ( 2 * upsilon11 );
    c12t = -c12t - u01' * C02 - c01' * U02 ...
                 - gamma11 * u12t;
    c12t = SolveSylv( upsilon11, U22, c12t );
    %--------------------------------------------%
    [ CTL, CTR, ...
      CBL, CBR ] = FLA_Cont_with_3x3_to_2x2( ...
                        C00,  c01,       C02,  ...
                        c10t, gamma11, c12t, ...
                        C20,  c21,       C22, 'FLA_TL' );
    [ UTL, UTR, ...
      UBL, UBR ] = FLA_Cont_with_3x3_to_2x2( ...
                        U00,  u01,       U02,  ...
                        u10t, upsilon11, u12t, ...
                        U20,  u21,        U22, 'FLA_TL' );
  end
  C_out = [ CTL, CTR
            CBL, CBR ];
return
```

Fig. 8. M-script implementation of unblocked Variant 3.

the computation, and then scaling it by dividing it by 10^9. The commands in the body of the loop map are implemented as calls to Basic Linear Algebra Subprograms (BLAS) [LHKK79, DDCHH88, DDCHD90], an interface to commonly encountered linear algebra operations, as well as other routines supported by libflame, which themselves call BLAS operations. As part of our project, we have derived a full library of these operations, but for this experiment we are depending on optimized implementations provided by the GotoBLAS2 implementation [GvdG08b, GvdG08a]. For the blocked algorithms, a block size of 128 was used. While most of our papers focus on achieving high performance, this paper does not and hence we do not go into further details regarding the experiments.

The important thing to note is that distinct variants give rise to different performance and that therefore there is a benefit to identifying all algorithmic variants. For other operations, targeting sequential, multi-threaded, and distributed memory architectures, we have consistently shown the benefit of having a choice of algorithms. Several illuminating examples, including derivations and performance comparisons, can be found in [GGHvdG01, BGM+05, QOvdG03, vdGQO08, BGdG].
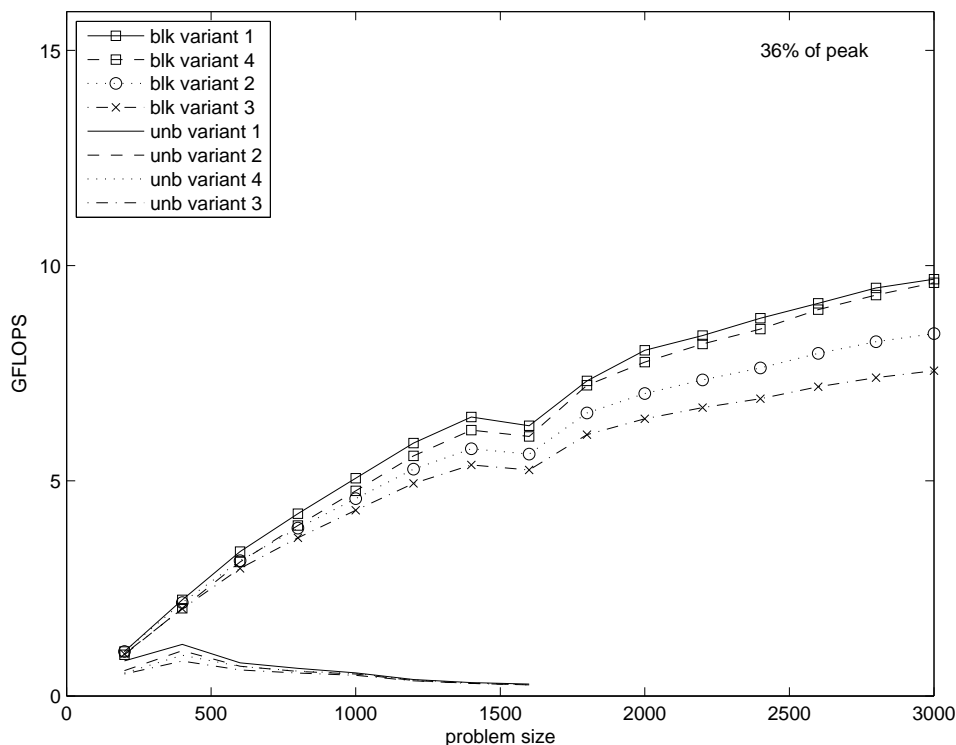
Fig. 9. Performance of the various algorithms, on a four core architecture.

4. Past, Current, and Future Directions

This paper gives a refined presentation of the methodology already proposed in [BGM+05, vdGQO08]. In this section, we discuss the developments that this work has enabled over the last decade, the current impact of the project, and future directions.

Mechanical derivation of linear algebra libraries. Early on in the project it was recognized that with the introduction of the worksheet, the methodology became systematic to the point where it could be automated. This led to a mechanical system, implemented in Mathematica, that does just that [Bie06]. For an operation like the solution of a triangular Lyapunov equation, that system performs the steps described in Section 2 mechanically, provided that a loop-invariant is given by the user. The system outputs an algorithm, M-script implementation, and C implementation. More recently, a system that can automatically generate PMEs has been developed [FTB11b], as well as one that then automatically generates loop-invariants [FTB11a]. Combining these efforts would yield a system that, given a linear algebra operation, would automate all steps described in this paper.

Scope. As part of the `libflame` library, the illustrated methodology has been applied to all operations supported by the BLAS and a large number of operations supported by the LAPACK library. In all, the `libflame` library, written in C, comprises about 1500 implementations of algorithms for about 150 distinct operations (operations that differ in whether they work, for example, with upper or lower triangular matrices are counted separately). Most of these were formally derived as described in this paper.

Formal derivation of stability analyses. The notion of correctness of an algorithm is somewhat different in the presence of round-off error. There, we must instead establish that an algorithm is numerically stable. We have extended the derivation process and worksheet described in this paper so that once algorithms have been derived, numerical stability results can be systematically derived in a similar fashion [Bie06, BvdG11].

Sparse linear solvers. As mentioned in the introduction, sparse iterative solvers for linear algebra problems are more commonly used by applications. These are based on Krylov subspace methods, the Conjugate Gradient method being one example. Recently, we showed that the formal derivation methodology we developed can be extended to the formal derivation of these Krylov subspace methods [EBvdG10].

5. Conclusion

We have demonstrated how formal derivation of loop-based algorithms is both viable and valuable for the domain of dense linear algebra. Key to the success of the methodology has been a notation that hides indexing details when using arrays and subarrays, the definition of an operation to be implemented via the Partitioned Matrix Expression (PME), a systematic way for identifying loop-invariants, and a framework (the worksheet) that captures the Fundamental Invariance Theorem in a way that clearly links it to a loop-based algorithm.

Additional information

In this document, we cite a large number of our own papers. This is intended to convey how broadly applicable the methodology is and the extend of its practical impact. This, however, obscures what papers are particularly important for a reader who wants to learn more. We recommend first reading the paper in which the worksheet was initially proposed [BGM^{+}05] and the book that we wrote for a general audience, at the undergraduate level [vdGQO08]. The FLAME project website, `http://www.cs.utexas.edu/users/flame/` is also a useful resource.

References

[ABB^{+}99]   E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (Third ed.)*. Philadelphia, PA, USA, 1999.

[BGdG]   Paolo Bientinesi, Brian Gunter, and Robert A. Van de Geijn. Families of algorithms related to the inversion of a symmetric positive definite matrix. *ACM Trans. Math. Soft.*, 35(1).

[BGM^{+}05]   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 Trans. Math. Soft.*, 31(1):1–26, March 2005.

[Bie06]   Paolo Bientinesi. *Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, The University of Texas, 2006. Technical Report TR-06-46. September 2006.

[BQOvdG05] Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application programming interfaces. *ACM Trans. Math. Soft.*, 31(1):27–59, March 2005.

[BvdG11]   Paolo Bientinesi and Robert A. van de Geijn. Goal-oriented and modular stability analysis. *SIAM Journal on Matrix Analysis and Applications*, 32(1):286–308, 2011.

[DDCHD90] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[DDCHH88] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[Dij68]   E. W. Dijkstra. A constructive approach to the problem of program correctness. *BIT*, 8:174–186, 1968.

[Dij76]   E. W. Dijkstra. *A discipline of programming*. Prentice Hall, 1976.

[EBvdG10] Victor Eijkhout, Paolo Bientinesi, and Robert van de Geijn. Towards mechanical derivation of Krylov solver libraries. *Procedia Computer Science*, 1(1):1799 – 1807, 2010. Proceedings of ICCS 2010, `http://www.sciencedirect.com/science/publication?issn=18770509&volume=1&issue=1`.

[FTB11a]   Diego Fabregat-Traver and Paolo Bientinesi. Automatic generation of loop-invariants for matrix operations. In *Proceedings of the 11th International Conference on Computational Science and its Applications*, 2011. To appear. Also TR AICES-2010/02-1, AICES, RWTH Aachen.

[FTB11b]   Diego Fabregat-Traver and Paolo Bientinesi. Knowledge-based automatic generation of partitioned matrix expressions. In *Proceedings of the 13th International Workshop on Computer Algebra in Scientific Computing*, 2011. To appear. Also TR AICES-2010/01-3, AICES, RWTH Aachen.

[GGHvdG01] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.

[Gri81]   David Gries. *The Science of Programming*. Springer-Verlag, 1981.

[Gun01]   John A. Gunnels. *A Systematic Approach to the Design and Analysis of Parallel Dense Linear Algebra Algorithms*. PhD thesis, Department of Computer Sciences, The University of Texas, December 2001.

[GvdG01]   John A. Gunnels and Robert A. van de Geijn. Formal methods for high-performance linear algebra libraries. In Ronald F. Boisvert and Ping Tak Peter Tang, editors, *The Architecture of Scientific Software*, pages 193–210. Kluwer Academic Press, 2001.

[GvdG08a] Kazushige Goto and Robert van de Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Soft.*, 35(1):1–14, 2008.

[GvdG08b] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Soft.*, 34(3: Article 12, 25 pages), May 2008.

[Hoa69]   C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, pages 576–580, October 1969.

[Kha02]   Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002.

[LHKK79]   C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.

[MLB87]   C. Moler, J. Little, and S. Bangert. *Pro-Matlab, User's Guide*. The Mathworks, Inc., 1987.

[PvdGB]   Jack Poulson, Robert van de Geijn, and Jeffrey Bennighof. Parallel algorithms for reducing the generalized Hermitian-definite eigenvalue problem. *ACM Trans. Math. Soft.* submitted.

[QOvdG03] Enrique S. Quintana-Ortí and Robert A. van de Geijn. Formal derivation of algorithms: The triangular Sylvester equation. *ACM Trans. Math. Soft.*, 29(2):218–243, June 2003.

[vdGQO08] Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `http://www.lulu.com/content/1911788`, 2008.

[VZCvdG⁺09] Field G. Van Zee, Ernie Chan, Robert van de Geijn, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. Introducing: The libflame library for dense matrix computations. *IEEE Computation in Science & Engineering*, 11(6):56–62, 2009.

[Zee09]   Field G. Van Zee. `libflame`: *The Complete Reference*. `www.lulu.com`, 2009.