Deriving Linear Algebra Libraries

FLAME Working Note #57

Robert van de Geijn
Tyler Rhodes
Maggie Myers
Field G. Van Zee

Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{rvdg,trhodes,myers,field}@cs.utexas.edu

March 13, 2011

Abstract

Starting in the late 1960s computer scientists including Dijkstra and Hoare advocated goal-oriented programming and formal derivation of algorithms. The problem was that for loop-based programs, *a priori* determination of loop-invariants, a prerequisite for developing loops, was a task too complex for any but the simplest of operations. We believe that no practical progress was made in the field until around 2000 when we discovered how to systematically determine loop-invariants for operations in 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. It has yielded a system for mechanical derivation of algorithms and a high-performance linear algebra library, `libflame`, that is largely derived to be correct and includes more than a thousand algorithmic variants of algorithms for more than a hundred linear algebra operations. To our knowledge, this success story has unfolded without any awareness on the part the formal methods community. This paper is meant to raise that awareness.

1 Introduction

The domain of linear algebra libraries is at the bottom of the food chain of scientific computing. While most practical applications give rise to sparse linear algebra problems that are solved via so-called iterative methods that converge to a solution, a significant number of them spend most computational time solving dense matrix problems. Even sparse linear algebra problems often have dense subproblems to be solved. As a result, LAPACK [1], a package for dense matrix operations, developed in the late 1980s and early 1990s and still programmed in Fortran-77, 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 modern replacement of LAPACK, `libflame` [26]. The domain poses a few interesting challenges: scientific computing tends to exploit the latest architectures and of those architectures demands the highest performance possible. This requires the design of loop-based algorithms that cast most computation in terms of high-performing matrix-matrix computations (like matrix-matrix multiplication). The loop steps through matrices with a block size chosen so as to optimize the reuse of data in caches. For a specific operation, there are often a number of algorithmic variants, with one algorithmic variant matching a given architecture better than the others, thus yielding high performance.

Step	**Annotated Algorithm:** $[D, E, \ldots] := \mathrm{op}(A, B, \ldots)$
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	$\{P_{post}\}$

Figure 1: Blank FLAME worksheet used to derive algorithms

Thus, a library should incorporate all loop-based algorithms for a given operation so that the best one can be chosen, which LAPACK does not. This brings up the question of how to systematically find all algorithmic variants for a given operation. Formal derivation of loops turns out to be the answer [17, 15, 16, 3, 23, 2, 24, 4], as we will illustrate in this paper.

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 [14], which itself is based on the works of Dijkstra [7, 6] and Hoare [18]. What the paper does provide is what we believe to be an excellent practical example of the application of formal derivation of loops.

2 Derivation of Linear Algebra Algorithms

In this section, we walk the reader through the derivation process. This section is completely routine for us, having been applied 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 needs not worry: the methology is systematic to the point where one needs not be an expert to apply it. A treatment that targets novices and has been used at the undergraduate level can be found in [24].

2.1 The FLAME methodology

A fundamental insight in our project was the realization that the Fundamental Invariance Theorem, used to prove the correctness of a loop in a program, can be formulated as a *worksheet* that is systematically filled, first with assertions (predicates) and then with commands (imperative statements) [3]. The worksheet is given in Figure 1. There, grey and white areas will be filled with predicates that indicate the prescribed state and commands, respectively. It is filled 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-invariant, 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 for a prototypical example.

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

Figure 2: Worksheet for deriving the unblocked algorithm corresponding to loop-invariant 3.

2.2 Example: the solution of the triangular Lypunov equation

We now show how the methodology is applied to a prototypical example: the solution of the triangular Lyapunov equation given by $U^T X + XU = -C$, where U is upper triangular and C is a symmetric matrix so that only the upper triangular part needs to be stored. Here ^{T} indicates transposition.

The solution X, which is also symmetric, is to be computed and will overwrite the upper triangular part of C. This operation is preceded by a pre-process that takes the general time-continuous Lyapunov equation (an operation encountered in control theory [19]) to the given triangular Lyapunov form.

2.3 Filling the worksheet

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

Step 1: The precondition and postcondition.

Letting $\hat{C}$ denote the original contents of C, the precondition is given by $C = \hat{C}$ while the postcondition is $C = X \wedge U^T X + XU = -\hat{C}$. Here the $\hat{}$ is needed to be able to reason about the current contents (state) of variable C relative to the original contents, $\hat{C}$.

Step 2: Deriving the loop-invariants. A fundamental insight is that algorithms sweep through matrices (arrays) in a systematic fashion. Since all operands are either triangular or symmetric, we will partition all into quadrants, since this allows regions of the matrices that are either zero, for the triangular matrix, or not stored, for the symmetric matrices, to be exposed:

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

where U_{TL}, X_{TL}, C_{TL}, and $\hat{C}_{TL}$ are all conformal (of same size) and square. The 0 and $\star$ indicate a submatrices that are entirely zero or not stored, respectively.

Substituting these into the postcondition yields the expression in Figure 3, which we call the *Partitioned Matrix Expression* (PME). It is a recursive definition of the operation in terms of the exposed submatrices. **Partitioning of the operands, substituting these into the postcondition, and applying the rules of linear algebra yields the PME.**

The PME expresses *all* computation that must have occurred to compute the operation in terms of the exposed quadrants. The observation is that as long as the loop has not finished, only *some* of the PME is satisfied. Thus, to come up with loop-invariants, one deletes some of the subexpressions in the PME, as illustrated in Figure 4. **Invariants are thus systematically derived from the PME.**

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

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

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

implies $U^T X + XU = -C$. **This dictates the (nonunique) choice for the loop-guard:** $G = (m(U_{BR}) < m(U))$, where $m(\cdot)$ returns the row dimension of the indicated matrix.

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 where the top-left quadrants are 0×0 (empty).**

Step 5: Moving through the matrices. In Steps 5a and 5b, submatrices are exposed so that we move forward 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 movement through the matrices. Greek letters denote scalars, lower case letters column vectors, and upper case letter matrices. Submatrices like u_{12}^T can be easily recognized as being part of a row and hence a row vector (transposed column vector).

Substituting the partitioned operands 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

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

Figure 3: The Partitioned Matrix Expression (PME) for the solution of the triangular Lyapunov equation.

Loop-invariant 1:	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ \cancel{U_{TL}^T} X_{TR} \cancel{+ X_{TR} U_{BR}} = \hat{C}_{TR} \cancel{- X_{TL} U_{TR}} \\ \cancel{U_{BR}^T} X_{BR} \cancel{+ X_{BR} U_{BR}} = \hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{cases}$
Loop-invariant 2:	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ \cancel{U_{TL}^T} X_{TR} \cancel{+ X_{TR} U_{BR}} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T} X_{BR} \cancel{+ X_{BR} U_{BR}} = \hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{cases}$
Loop-invariant 3:	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T} X_{BR} \cancel{+ X_{BR} U_{BR}} = \hat{C}_{BR} \cancel{- (U_{TR}^T X_{TR} + X_{TR}^T U_{TR})} \end{cases}$
Loop-invariant 4:	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline \star & C_{BR} \end{array}\right) = \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline \star & X_{BR} \end{array}\right) \wedge \begin{cases} U_{TL}^T X_{TL} + X_{TL} U_{TL} = -\hat{C}_{TL} \\ U_{TL}^T X_{TR} + X_{TR} U_{BR} = -\hat{C}_{TR} - X_{TL} U_{TR} \\ \cancel{U_{BR}^T} X_{BR} \cancel{+ X_{BR} U_{BR}} = -\hat{C}_{BR} - (U_{TR}^T X_{TR} + X_{TR}^T U_{TR}) \end{cases}$

Figure 4: Loop-invariants for the triangular Lyapunov equation.

The fact that the top-left quadrant starts empty and must eventually envelop the entire matrix dictates how the algorithm moves through the matrices. The movement through the matrix, together with the finite size of the operands, means that there is a natural loop-bound function: $t = n - m(U_{TL})$ that is decreased every time through the loop.

Step 6: State before the update. The commands in Step 5a are merely indexing operations. Since no computation happens 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

$$U_{TL} \rightarrow U_{00}, U_{TR} \rightarrow \left(\begin{array}{c|c} u_{01} & U_{02} \end{array} \right), \text{ etc.}$$

and linear algebra manipulation as illustrated in Figure 6. This yields the state described by Step 6. **The invariant together with the repartitioning 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 in the loop-invariant

$$U_{TL} \rightarrow \left(\begin{array}{c|c} U_{00} & u_{01} \\ \hline 0 & \upsilon_{11} \end{array} \right), U_{TR} \rightarrow \left(\begin{array}{c} U_{02} \\ \hline u_{12}^T \end{array} \right), \text{ etc.}$$

and linear algebra manipulation. **The invariant and the redefinition of the quadrants in Step 5b dictate the predicate in Step 7** via a process much like that illustrated in Figure 6.

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:

```
Algorithm: $C :=$ LYAP_UNB$(U, C)$
Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $X \rightarrow \cdots$
    where $U_{TL}$ is $0 \times 0$, $X_{TL}$ is $0 \times 0$, $C_{TL}$ is $0 \times 0$
while $m(U_{TL}) < m(U)$ do
    Repartition
        $\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$
        where $\upsilon_{11}, \chi_{11}, \gamma_{11}$ are scalars

    Variant 1
        $c_{01} := -c_{01} - C_{00}u_{01}$
        Solve $U_{00}^T c_{01} + c_{01}\upsilon_{11} = c_{01}$
        $\gamma_{11} := -\gamma_{11} - u_{01}^T c_{01} - c_{01}^T u_{01}$
        $\gamma_{11} := \gamma_{11}/(2\upsilon_{11})$
    Variant 2
        Solve $U_{00}^T c_{01} + c_{01}\upsilon_{11} = c_{01}$
        $\gamma_{11} := -\gamma_{11} - u_{01}^T c_{01} - c_{01}^T u_{01}$
        $\gamma_{11} := \gamma_{11}/(2\upsilon_{11})$
        $C_{02} := C_{02} - c_{01}u_{12}^T$
        $c_{12}^T := -c_{12}^T - \gamma_{11}u_{12}^T - c_{01}^T U_{02}$
    Variant 3
        $\gamma_{11} := -\gamma_{11} - 2u_{01}^T c_{01}$
        $\gamma_{11} := \gamma_{11}/(2\upsilon_{11})$
        $c_{12}^T := -c_{12}^T - u_{01}^T C_{02} - c_{01}^T U_{02} - \gamma_{11}u_{12}^T$
        Solve $\upsilon_{11}c_{12}^T + c_{12}^T U_{22} = c_{12}^T$
    Variant 4
        $\gamma_{11} := \gamma_{11}/(2\upsilon_{11})$
        $c_{12}^T := c_{12}^T - \gamma_{11}u_{12}^T$
        Solve $\upsilon_{11}c_{12}^T + c_{12}^T U_{22} = c_{12}^T$
        $C_{22} := C_{22} - u_{12}c_{12}^T - c_{12}u_{12}^T$

    Continue with
        $\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$
endwhile
```

```
Algorithm: $C :=$ LYAP_BLK$(U, C)$
Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $X \rightarrow \cdots$
    where $U_{TL}$ is $0 \times 0$, $X_{TL}$ is $0 \times 0$, $C_{TL}$ is $0 \times 0$
while $m(U_{TL}) < m(U)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\cdots$
        where $\upsilon_{11}, \chi_{11}, \gamma_{11}$ are scalars

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

Figure 5: Algorithms for computing the solution to the triangular Lypunov equation. Left: unblocked algorithms. Right: blocked algorithms.

- C_{00} already contains X_{00} and needs not be updated.
- c_{01} already contains x_{01} and needs not be updated.
- γ_{11} holds $\hat{\gamma}_{11}$ and needs to be overwritten by the solution of $2u_{01}^T x_{01} + 2\upsilon_{11}\chi_{11} = -\hat{\gamma}_{11}$. Recognizing that by this step c_{01} holds x_{01} and γ_{11} holds $\hat{\gamma}_{11}$, this can be accomplished by updating γ_{11} with
$$\gamma_{11} = (-\gamma_{11} - 2u_{01}^T c_{01})/(2\upsilon_{11}).$$

Substituting the repartitioning in Step 5a (which hides indexing)

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

which stands for

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

into the state of the variables at the top of the loop (the invariant)

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

yields

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

which, after algebraic manipulation using the rules of linear algebra, yields the expression in Step 6.

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

- And so forth.

Resulting algorithm. The resulting algorithm, stripped of the annotations that were used to derive it, is given in Figure 5 (left), executing only the commands indicated under Variant 3.

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 at time. All resulting algorithms are given in Figure 5.

2.5 Discussion

It is the notation we use that enables the derivation process: by presenting submatrices rather than index ranges, the derived algorithm avoids much of the indexing clutter that is typically found in conventional loop-based algorithms. Indeed, the algorithm exposes only one loop even thought the algorithm requires approximately n^3 floating point operations. Where are the other loops? 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 to be correct.

2.6 From algorithm to code

Having a correct algorithm does not mean one has a correct implementation. To preserve the correctness of the algorithm as we translate it to code, we defined APIs for different languages so that the code closely

```
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
```

Figure 7: M-script implementation of unblocked Variant 3.

resembles the algorithm [5, 24, 25]. An example of this is given in Figure 7. In that figure, unblocked Variant 3 is coded in M-script, the scripting language of Matlab [21].

We created a handfull 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 the figure actually executes as-is, and gave the correct answer the first time it was executed. We have similar APIs for the C programming language [5, 24, 25, 26] and for distributed memory architectures, the recently developed Elemental library uses a similar API [22].

3 Performance

In Figure 8 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. While most of our papers are mostly about performance, this paper is not. 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, and dividing it by the time required to complete the computation. The important thing to note is that different variants give rise to different performance and that therefore there is a benefit to identifying all algorithmic variants. For other operations, targeting sequential, multi-

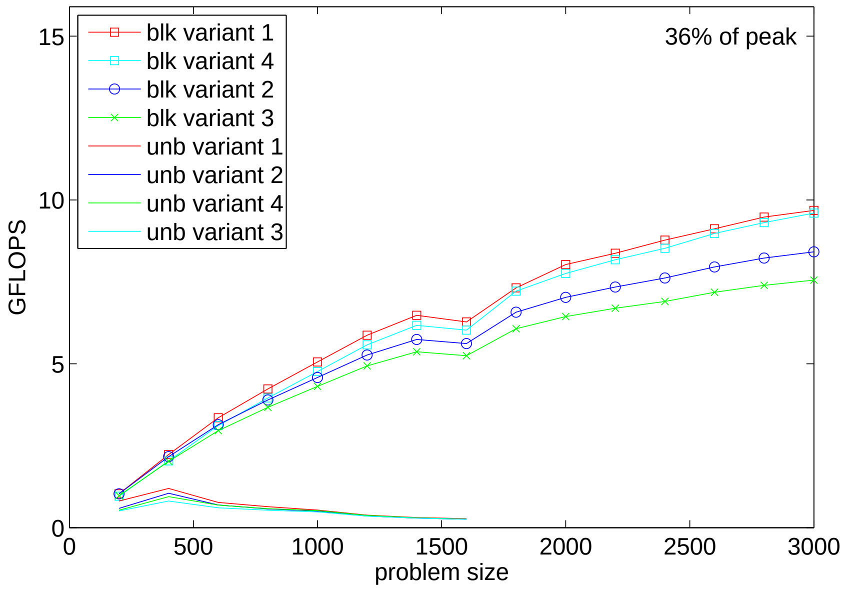

Figure 8: Performance of the various algorithms on a four core architecture.

threaded, and distributed memory architectures, we have consistently shown the benefit of having multiple algorithms available. Excellent examples, including derivations and performance comparisons, can be found in [16, 3, 23, 24, 4].

The commands in the body of the loop are calls to multithreaded Basic Linear Algebra Subprograms (BLAS) [20, 9, 8], an interface to commonly encountered linear algebra 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 of the BLAS [13, 12] For the blocked algorithms, a block size of 128 was used.

4 Past, Current, and Future Directions

This paper gives a refined presentation of the methodology already proposed in [3, 24]. In this section, we discuss 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 an implementation of a mechanical system, implemented in Mathematica, that does just that [2]. For an operation like the solution of a Lyapunov equation, that system can perform the steps described in Section 2 mechanically, provided 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 [11], which means that automatically generating loop-invariants is a relatively easy next step. 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 described methology has been applied to all operations supported by the BLAS and a large number of operations supported by the LAPACK library. In all, this C library comprises about 1500 implementations of algorithms for about 150 distinct operations (where operations that differ in whether they work, for example, with upper or lower triangular matrices are counted separately). Most of these were derived to be correct, although for some we did so somewhat less formally than described in this paper.

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 so-called Krylov subspace methods. (The Conjugate Gradient method is an example.) Recently, we showed that the formal derivation methodology we developed can be extended to the formal derivation of these Krylov subspace methods [10].

5 Conclusion

In this paper, we have demonstrated how formal derivation of loop-based algorithms is viable and valuable for the domain of dense linear algebra. Key to success 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.

Acknowledgments

This research was initially started as a Freshman Research Initiative project that also involved Eileen Martin, Burns Healey, and Nick Wiz. We thank Jay Misra for encouraging us to target the formal methods community with this paper and the remaining members of the FLAME team for their contributions over the last ten years.

This research was supported in part by NSF grants CCF-0342369, CCF-0540926, CCF-0702714, and CCF-0917167, and Microsoft.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] 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.

[2] 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.

[3] 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*, 31(1):1–26, March 2005.

[4] Paolo Bientinesi, Brian Gunter, and Robert A. Van de Geijn. Families of algorithms related to the inversion of a symmetric positive definite matrix. *ACM Transactions on Mathematical Software*, 35(1).

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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`.

[11] Diego Fabregat and Paolo Bientinesi. Automatic generation of partitioned matrix expressions for matrix operations. FLAME Working Note #51 AICES-2010/10-1, Aachen Institute for Computational Engineering Science, RWTH Aachen, October 2010.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

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

[22] Jack Poulson, Robert van de Geijn, and Jeffrey Bennighof. Parallel algorithms for reducing the generalized hermitian-definite eigenvalue problem. *ACM Transactions on Mathematical Software*. submitted.

[23] 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.

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

[25] 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.

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