Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2011/01-3

16/January/2011

Knowledge-Based Automatic Generation of Partitioned Matrix Expressions

D. Fabregat-Traver and P. Bientinesi

RWTH AACHEN UNIVERSITY

AICES

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.

List of AICES technical reports: http://www.aices.rwth-aachen.de/preprints

Knowledge-Based Automatic Generation of Partitioned Matrix Expressions

Diego Fabregat-Traver Paolo Bientinesi
AICES, RWTH Aachen
52062, Aachen, Germany
{fabregat,pauldj}@aices.rwth-aachen.de

ABSTRACT

Our objective is the automatic generation of algorithms and routines for matrix equations. We aim for a fully automated system that from the sole description of a target equation creates multiple algorithms without any human intervention. We achieve automation through a methodology based on formal methods and program correctness. The methodology consists of three main stages. The first stage yields the core object of the process, the Partitioned Matrix Expression (PME), a decomposition of the target problem into simpler sub-problems. In the second stage the PME is used to identify predicates, the Loop Invariants, that are used to set up the skeleton of a family of proofs of correctness. In the third and last stage the actual algorithms are then constructed so that they satisfy the proof of correctness. In this paper we focus on the first stage of the process, the automatic generation of Partitioned Matrix Expressions. In particular, we discuss the steps leading to a PME and the knowledge necessary for a symbolic system to perform such steps. We also introduce Cl1ck, a prototype system that generates PMEs automatically.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algebraic algorithms; D.1.2 [**Programming Techniques**]: Automatic Programming

General Terms

Automation, Algorithms

Keywords

Partitioned Matrix Expression, Algorithm Generation, Pattern Matching, Rewrite Rules

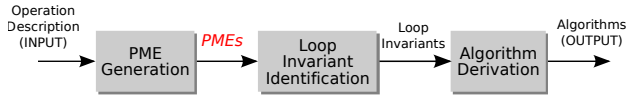

Figure 1: The three main stages in the process of derivation of algorithms.

1. INTRODUCTION

The computing landscape offers an amazing variety of architectures and programming paradigms[1], each one affecting heavily performance and possibly even accuracy of numerical algorithms. In order to attain high-performance in such diverse scenarios, for every target problem not one but many different algorithms are needed [3]. We envision the automatic generation of such variety of algorithms. Our focus is on matrix equations: we aim at a symbolic system that takes as input the description of a target equation Eq and—without any human intervention—returns a family of algorithms that solve Eq.

One of the authors presented a methodology, based on formal methods and program correctness, to derive families of algorithms [1]. The derivation process, consisting of three main stages, is systematic and entirely determined by the mathematical description of the input equation [2]. The process is depicted in Fig. 1. The input is the description of a target equation. In the first stage, the *Partition Matrix Expression* (PME) for the equation is obtained. A PME is a matrix-like object that exposes how an operation can be decomposed into simpler sub-problems. An example is shown in Box 1. Once the PME is available, the second stage of the

$$\left(\frac{X_T = \Omega(L_{TL}, U, C_T)}{X_B = \Omega(L_{BR}, U, C_B - L_{BL}X_T)} \right)$$

Box 1: Partitioned Matrix Expression for the triangular Sylvester equation.

[1]Current processors comprise between 2 and 40 cores with frequencies from 800 up to 3000 MHz, include between 1 and 3 levels of (shared) cache, and rely on memories ranging from 2 to dozens of GBs; they may be found in combination with one or more GPUs, and assembled in flat or hierarchical fashion to form clusters and supercomputers. The main programming paradigms for such diverse architectures include message passing, multi-threading, data parallelism, and hybrid approaches.

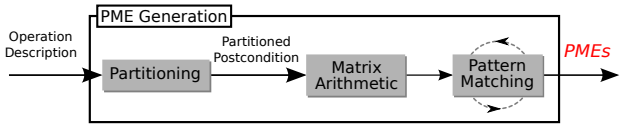

Figure 2: Steps for the automatic generation of PMEs.

process deals with the identification of boolean predicates, the *Loop-Invariants* [4], that describe the intermediate state of computation for the sought-after algorithms. In the third and last stage, each Loop-Invariant is used to set up a proof of correctness around which the algorithm is finally built.

This paper centers around the first stage of the derivation process, the generation of PMEs; the objective is automation, in the form of a symbolic system. First, we identify the minimum amount of knowledge about the operation required by a system to perform all the subsequent stages automatically. We introduce a formalism to input into the system the target equation along with the auxiliary knowledge. We then describe the process for transforming an input equation into PMEs. As Fig. 2 shows, such process involves three steps: 1) the partitioning of the operands in the equation, 2) matrix arithmetic involving the partitioned operands, and 3) a sequence of iterations, each consisting of algebraic manipulation and pattern matching. We demonstrate that the process can indeed be automated through CLICK[2], a symbolic system that performs all the steps involved in the PME generation.

The paper is organized as follows. In Section 2 we categorize the input needed by a symbolic system. Partitionings of the operands and inheritance of properties are discussed in Section 3.1, while in Section 3.2 we describe how to use partitionings to obtain PMEs. We draw conclusion in Section 4.

2. INPUT TO THE SYSTEM

Our first concern is to establish how a target operation should be formally described. We use the Cholesky factorization as an example: given a symmetric positive definite (SPD) matrix A, the goal is to find a lower triangular matrix L such that $LL^T = A$. We ask two questions: What is a suitable description of the operation? Does such a description provide enough information for a symbolic system to carry out all the derivation stages? The most common formula to identify the Cholesky factorization is

$$LL^T = A; \tag{1}$$

the equation is self-explanatory to anyone who has familiarity with numerical linear algebra: a) the letter L is used to denote lower triangular matrices, b) the unknown quantity lays in the left-hand side of Eq. (1), c) the known quantity is in the right-hand side, and d) the existence of the solution is assumed implicitly.

Even if it were explicitly stated that L is lower triangular, A is SPD, and their size matches, the specification would still not identify a Cholesky factorization unequivocally. For example, what happens if the left and right-hand sides of Eq. (1) are flipped?

$$A = LL^T \tag{2}$$

[2]The name CLICK epitomizes the idea that the effort a user has to make to obtain algorithms consists of just *one click*.

Is Eq. (2) still a representation of the Cholesky factorization of A, or it instead represents a matrix product in which A is the unknown, and the lower triangular matrix L is multiplied by its transpose? No definitive answer can be given unless each operand is labeled as known (input) or unknown (output).

Let us now consider a formula representing a simple generalization of Eq. (2):

$$XY = Z. \tag{3}$$

Depending on which of the operands X, Y and Z are known, the formula represents a matrix-matrix multiplication, the solution of a system of equations, a matrix factorization, or a tautology. In Table 1 we relate the assumptions made for each of the operands to the possible interpretations of Eq. (3).

Although simple, the example already leads to a number of observations. As humans, in describing an equation we rely on implicit knowledge and notational conventions (letters, position, properties). However, our notation is often ambiguous. Since we are aiming for a fully-automated system, i.e., without any human intervention, we need a formalism to unequivocally describe a target equation. We choose the language traditionally used to reason about program correctness: equations shall be specified by means of the predicates Precondition (P_{pre}) and Postcondition (P_{post}) [4]. The precondition enumerates the operands that appear in the equation and describes their properties, while the postcondition specifies the equation to be solved.

Box 2 contains the description of the Cholesky factorization; the notation $L = \Gamma(A)$ indicates that L is the Cholesky factor of A. Even though the definition is unambiguous, it

$$L = \Gamma(A) \equiv \begin{cases} P_{\text{pre}} : \{\texttt{Unknown}(L) \wedge \texttt{LowerTriangular}(L) \wedge \\ \qquad \texttt{Known}(A) \wedge \texttt{SPD}(A)\} \\ P_{\text{post}} : \{LL^T = A\} \end{cases}$$

Box 2: Formal description for the Cholesky factorization.

does not include all the information needed by a symbolic system to fully automate the derivation process. In Sec. 2.1 we discuss how a system expands its knowledge by "learning of" new equations, and in Sec. 3 we overview the ground knowledge that a system must possess relative to matrix partitioning and inheritance of properties.

2.1 Pattern Learning

We refer to the pair of predicates in Box 2 as the pattern that identifies the Cholesky factorization. Such a pattern establishes that matrices L and A are one the Cholesky factor of the other provided that the constraints in the precondition are satisfied, and L and A are related as dictated in the postcondition ($LL^T = A$). For instance, in the expression

$$XX^T = A - BC,$$

in order to determine whether $X = \Gamma(A - BC)$, the following facts need to be asserted: i) X is an unknown lower triangular matrix; ii) the expression $A - BC$ is a known quantity (A, B and C are known); iii) the matrix $A - BC$ is symmetric positive definite.

Operation	X	Y	Z
Matrix-matrix multiplication	known	known	unknown
Linear system (1)	unknown	known, non-singular	known
Linear system (2)	known, non-singular	unknown	known
Cholesky factorization	unknown, lower triangular,	transpose of X	known, SPD
LU decomposition	unknown, lower triangular, unit diagonal	unknown, upper triangular	known, $\exists\ LU(Z)$
QR decomposition	unknown, orthonormal	unknown, upper triangular	known
Tautology	known	known	known

Table 1: Relation between properties of the operands and the interpretation of $XY = Z$.

The strategy for decomposing an equation in terms of simpler problems greatly relies on pattern matching. In the next section we describe how matrix equations can be rewritten in terms of sub-matrices, resulting in expressions seemingly more complicated than the initial formulation. Such expressions are thus inspected to find segments corresponding to known patterns.

Our system CLICK initially only knows the patterns for a basic set of operations: addition, multiplication, inversion, and transposition of matrices, vectors and scalars. This information is hard-coded. More complex patterns are instead discovered during the process of PME generation. For instance, the first time the PME for the Cholesky factorization is generated, CLICK learns and stores the pattern specified by Box 2. Thanks to such patterns it will then be possible to identify that a Cholesky factorization may be decomposed into a combination of triangular systems and simpler Cholesky factorizations. As CLICK's pattern knowledge increases, also does its capability of tackling complex operations.

3. FROM DESCRIPTIONS TO PMES

In this section we pedantically illustrate all the steps performed by CLICK to transform the description of the input equation into one or more PMEs. The idea is to first rewrite the postcondition in terms of partitioned matrices and then apply pattern matching to identify known operations. To this end, we introduce a set of rules to partition and combine operands and to assert properties of expressions involving sub-operands. The application of these rules to the postcondition yields a predicate called *partitioned postcondition*. From there, an iterative process consisting of algebraic manipulation and pattern matching takes us to the PMEs.

3.1 Partitioning and Inheritance

The discussion commences with a set of rules for partitioning matrices and vectors and for transferring properties to sub-matrices and sub-vectors. These rules are part of the basic engine of CLICK. Due to constraints imposed by both the structure of the input operands and the postcondition, only few partitioning rules will be admissible.

3.1.1 Operands partitioning and direct inheritance

As shown in Box 3, a generic matrix A can be partitioned in four different ways. The 1×1 rule (Box 3(d)) is special as it does not affect the operand; we refer to it as the *identity*. For a vector, only the 2×1 and 1×1 rules apply, while for scalars only the identity is admissible. When referring to any of the parts resulting from a non-identity rule, we use the terms sub-matrix or sub-operand, and for 2×2 partitionings we also use the term quadrant.

$$A_{m\times n} \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$$
where A_{TL} is $k_1 \times k_2$
(a) 2×2 rule

$$A_{m\times n} \rightarrow \left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right)$$
where A_T is $k_1 \times n$
(b) 2×1 rule

$$A_{m\times n} \rightarrow \left(A_L \,|\, A_R\right)$$
where A_L is $m \times k_2$
(c) 1×2 rule

$$A_{m\times n} \rightarrow \left(A\right)$$
where A is $m \times n$
(d) 1×1 (identity) rule

Box 3: Rules for partitioning a generic matrix operand A. We use the subscript letters T, B, L, and R for Top, Bottom, Left, and Right, respectively.

The inheritance of properties plays an important role in subsequent stages of the algorithm generation process. Thus, when the operands have a special structure, it is beneficial to choose partitioning rules that respect the structure. For a symmetric matrix, for instance, it is convenient to create sub-matrices that exhibit the same property. The 1×2 and 2×1 rules break the structure of a symmetric matrix, as neither of the two sub-matrices inherit the symmetry. Therefore, we only allow 1×1 or 2×2 partitionings, with the extra constraint that the TL quadrant has to be square.

Box 4 illustrates the admissible partitionings for lower tri-

angular (L) and symmetric (M) matrices. On the left, the identity rule is applied and the operands remain unchanged. On the right instead, a constrained 2×2 rule is applied, so that some of the resulting quadrants inherit properties. For a lower triangular matrix L, both L_{TL} and L_{BR} are square and lower triangular, L_{TR} is zero, and L_{BL} is a generic matrix. For a symmetric matrix M, both M_{TL} and M_{BR} are square and symmetric, and $M_{BL} = M_{TR}^T$ (or vice versa $M_{TR} = M_{BL}^T$). Each matrix type allows specific partitioning rules and inheritance of properties; for triangular, diagonal, symmetric, and SPD matrices such knowledge is hard-coded into Cl1ck.

$$L_{m\times m} \rightarrow \left(L\right) \quad \text{where } L \text{ is } m \times m \qquad \text{or} \qquad L_{m\times m} \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \quad \text{where } L_{TL} \text{ is } k \times k$$

(a) Viable partitionings for a lower triangular matrix.

$$M_{m\times m} \rightarrow \left(M\right) \quad \text{where } M \text{ is } m \times m \qquad \text{or} \qquad M_{m\times m} \rightarrow \left(\begin{array}{c|c} M_{TL} & M_{BL}^T \\ \hline M_{BL} & M_{BR} \end{array}\right) \quad \text{where } M_{TL} \text{ is } k \times k$$

(b) Viable partitionings for a symmetric matrix.

Box 4: Partitioning rules for structured matrices.

3.1.2 Theorem-aware inheritance

Although frequent, direct inheritance of properties is only the simplest form of inheritance. Here we expose a more complex situation. Let A be an SPD matrix. Because of symmetry, the only admissible partitioning rules are the ones listed in Box 4(b); applying the 2×2 rule, we obtain

$$A_{m\times m} \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right), \quad \text{where } A_{TL} \text{ is } k \times k \tag{4}$$

and both A_{TL} and A_{BR} are symmetric. More properties about the quadrants of A can be stated. For example, it is well known that *if A is SPD, then every principal sub-matrix of A is also SPD*. As a consequence, the quadrants A_{TL} and A_{BR} inherit the SPD property. Moreover, it can be proved that given a 2×2 partitioning of an SPD matrix as in (4), the following matrices (known as Schur complements) are also symmetric positive definite:

- $A_{TL} - A_{BL}^T A_{BR}^{-1} A_{BL}$,
- $A_{BR} - A_{BL} A_{TL}^{-1} A_{BL}^T$.

The knowledge emerging from this theorem is hard-coded into Cl1ck. In Sec. 3.2 it will become apparent how this information is essential for the generation of PMEs.

3.1.3 Combining the partitionings

The admissible rules are now applied to rewrite the postcondition. Since in general each operand can be decomposed in multiple ways, not one, but many partitioned postconditions are created. As an example, in the Cholesky factorization (Box 2) both the 1×1 and 2×2 rules are viable for both L and A, leading to four different rewrite sets:

- Both L and A are partitioned in 1×1.
- L and A are partitioned in 1×1 and 2×2, respectively.
- L and A are partitioned in 2×2 and 1×1, respectively.
- Both L and A are partitioned in 2×2.

Table 2 contains the resulting four partitioned postconditions. It is apparent that some of the expressions in the fourth column are not algebraically well defined. Consequently, in addition to constraints on each individual operand, the rules need to be such that the partitioned operands can be combined together according to standard matrix arithmetic. For instance, in the expression $X + Y$, if the 2×1 rule is applied to matrix X, the $+$ operator imposes that the same rule is applied to Y too.

With reference to Table 2, the rules in the second and third rows lead to ill-defined partitioned postconditions, thus they should be discarded. The second row leads to an expression whose left-hand and right-hand sides are a 1×1 and a 2×2 object, respectively. The reverse is true in the third row. Despite leading to a well defined expression, the first row of the table should be discarded too, as the goal is a *Partitioned* Matrix Expression and it leads to an expression in which none of the operands has been partitioned. In light of these additional restrictions, the only viable set of rules for the Cholesky factorization is the one given in the last row of Table 2, with the additional constraint that the TL quadrants are square.

In summary, partitioning rules must satisfy both the constraints due the nature of the individual operands, and those due to the operators appearing in the postcondition. In the next section we detail the algorithm used by Cl1ck to generate only the viable sets of partitioning rules.

3.1.4 Automation

We show how Cl1ck performs the partitioning process automatically. The naive approach would be to exhaustively search among all the rules applied to all the operands, leading to a search space of exponential size in the number of operands. Instead, Cl1ck utilizes an algorithm that traverses once the tree that represents the postcondition in prefix notation and yields only the viable sets of partitioning rules.

The algorithm builds around two main ideas: 1) the properties of an operand impose restrictions on the viable rules; 2) the operators in the postcondition constraint the partitionings of their operands. The input to the algorithm is the predicates P_{pre} and P_{post} for a target operation. As an example we look at the triangular Sylvester equation

$$LX + XU = C,$$

defined using our formalism as in Box 5.

$$X = \Omega(L, U, C) \equiv \begin{cases} P_{\text{pre}} : \{\texttt{Known}(L) \wedge \texttt{LowerTriangular}(L) \wedge \\ \qquad\qquad \texttt{Known}(U) \wedge \texttt{UpperTriangular}(U) \wedge \\ \qquad\qquad \texttt{Known}(C) \wedge \texttt{Unknown}(X)\} \\ P_{\text{post}} : \{LX + XU = C\}. \end{cases}$$

Box 5: Formal description for the triangular Sylvester equation.

#	L	A	Partitioned Postcondition
1	$L \rightarrow (L)$	$A \rightarrow (A)$	$(L)\,(L)^T = (A)$
2	$L \rightarrow (L)$	$A \rightarrow \left(\begin{array}{c\|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right)$	$(L)\,(L)^T = \left(\begin{array}{c\|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right)$
3	$L \rightarrow \left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$	$A \rightarrow (A)$	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c\|c} L_{TL}^T & L_{BL}^T \\ \hline 0 & L_{BR}^T \end{array}\right) = (A)$
4	$L \rightarrow \left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$	$A \rightarrow \left(\begin{array}{c\|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c\|c} L_{TL}^T & L_{BL}^T \\ \hline 0 & L_{BR}^T \end{array}\right) = \left(\begin{array}{c\|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right)$

Table 2: Application of the different combinations of partitioning rules to the postcondition.

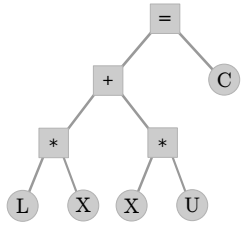

Figure 3: Tree representation of the equation $LX+XU = C$.

First, the algorithm transforms the postcondition to prefix notation (Fig. 3) and collects the name and the dimensionality of each operand. A list of disjoint sets, one per dimension of the operands is then created. This initial list for the Sylvester equation is

$$[\,\{L_r\},\{L_c\},\{U_r\},\{U_c\},\{C_r\},\{C_c\},\{X_r\},\{X_c\}\,]\,,$$

where r and c stand for *rows* and *columns* respectively. The algorithm traverses the tree, in a post-order fashion, to determine if and which dimensions are bound together. Two dimensions are bound to one another if the partitioning of one implies the partitioning of the other. If two dimensions are found to be bound, then their corresponding sets are merged together. As the algorithm moves from the leaves to the root of the tree, it keeps track of the dimensions of the operands' subtrees.

The algorithm starts by visiting the node corresponding to the operand L. There it establishes that the identity and the 2×2 partitioning rules are the only admissible ones. Thus, the rows and the columns of L are bound together, and the list becomes

$$[\,\{L_r, L_c\},\{U_r\},\{U_c\},\{C_r\},\{C_c\},\{X_r\},\{X_c\}\,]\,.$$

The next node to be visited is that of the operand X. Since X has no specific structure, its analysis causes no bindings. Then, the node corresponding to the $*$ operator is analyzed. The dimensions of L and X have to agree according to the matrix product, therefore, a binding between L_c and X_r is imposed:

$$[\,\{L_r, L_c, X_r\},\{U_r\},\{U_c\},\{C_r\},\{C_c\},\{X_c\}\,]\,.$$

At this stage the dimensions of the product LX are also determined to be $L_r \times X_c$.

The procedure continues by analyzing the subtree corresponding to the product XU. Again, the lack of a specific structure in X does not cause any binding and the algorithm follows with the study of the node for the operand U. The triangularity of U imposes a binding between U_r and U_c leading to

$$[\,\{L_r, L_c, X_r\},\{U_r, U_c\},\{C_r\},\{C_c\},\{X_c\}\,]\,.$$

Then, the node for the $*$ operator is analyzed, and a binding between X_c and U_r is found:

$$[\,\{L_r, L_c, X_r\},\{U_r, U_c, X_c\},\{C_r\},\{C_c\}\,]\,.$$

The dimensions of the product XU are determined to be $X_r \times U_c$.

The next node to be considered is the corresponding to the $+$ operator. It imposes a binding between the rows and the columns of the products LX and XU, i.e., between L_r and X_r, and between X_c and U_c. Since each of these pairs of dimensions already belong to the same set, no modifications are made to the list. The algorithm establishes that the dimensions of the $+$ node are $L_r \times U_c$. Next, the node associated to the operand C is analyzed. Since C has no particular structure, its analysis does not cause any modification. The last node to be processed is the equality operator $=$. This node binds the rows of C to those of L (C_r, L_r) and the columns of C to those of U (C_c, U_c). The final list consists of two separate groups of dimensions:

$$[\,\{L_r, L_c, X_r, C_r\},\{U_r, U_c, X_c, C_c\}\,]\,.$$

Having created g groups of bound dimensions, the algorithm generates 2^g combinations of rules (the dimensions within each group being either partitioned or not), resulting in a family of partioned postconditions, one per combination. In practice, since the combination including solely identity rules does not lead to a PME, only $2^g - 1$ combinations are acceptable. In our example the algorithm found two groups of bound dimensions, therefore three possible combinations of rules are generated: 1) only the dimensions in the second group are partitioned, 2) only the dimensions in the first group are partitioned, or 3) all dimensions are partitioned. The resulting partitionings are listed in Table 3.

This very same process is used to find the bound dimensions of every target operation and, accordingly, only each and every viable combination of partitioning rules is generated.

3.2 Matrix Arithmetic and Pattern Matching

This section covers the second and third steps in the PME generation stage (Fig. 2). Within the *Matrix Arithmetic*

#	L	U	C	X
1	(L)	$\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$	$\left(C_L \middle\| C_R\right)$	$\left(X_L \middle\| X_R\right)$
2	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$	(U)	$\left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right)$	$\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right)$
3	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right)$

Table 3: Viable combinations of partitioning rules for the Sylvester equation.

step, symbolic arithmetic is performed and the $=$ operator is distributed over the partitions, originating multiple equations. In Eq. 5 we display the result of these actions for the Cholesky factorization, where the symbol $\star$ means that the equation in the top-right quadrant is the transpose of the bottom-left one.

$$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c\|c} L_{TL}^T & L_{BL}^T \\ \hline 0 & L_{BR}^T \end{array}\right) = \left(\begin{array}{c\|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right) \quad \Rightarrow$$
$$\left(\begin{array}{c\|c} L_{TL}L_{TL}^T = A_{TL} & \star \\ \hline L_{BL}L_{TL}^T = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right). \qquad (5)$$

The *Pattern Matching* step delivers the sought-after PME. Success of this process is dependent on the ability to identify expressions with known structure and properties. In order to facilitate pattern matching, we force equations to be in their *canonical form*. We state that an equation is in canonical form if a) its left-hand side only consists of those terms that contain at least one unknown object, and b) its right-hand side only consists of those terms that solely contain known objects.

This last step carries out an iterative process comprising three separate actions: 1) structural pattern matching: equations are matched against known patterns; 2) once a known pattern is matched, the unknown operands are flagged as known and the equation becomes a tautology; 3) algebraic manipulation: the remaining equations are rearranged in canonical form. We clarify the iterative process by illustrating, action by action, how CLlCK works through the Cholesky factorization. The first iteration is depicted in Box 6, in which the top formula displays the initial state. In all the next expressions, green and red are used to highlight the known and unknown operands, respectively.

Structural pattern matching

All the equations in Box 6(a) are in canonical form. Through pattern matching, the top-left quadrant is the only one for which a match is found. CLlCK identifies the equation as a Cholesky factorization (Box 6(b)), since the pattern in Box 2 is satisfied.

Exposing new available operands

Having matched the top-left equation, CLlCK turns the unknown operand L_{TL} into L_{TL}, and propagates the information to all the other quadrants (Box 6(c)). As a result, the top-left equation becomes a tautology.

Algebraic manipulation

All the remaining equations are still in canonical form, thus no operation takes place (Box 6(d)).

$$\left(\begin{array}{c\|c} L_{TL}L_{TL}^T = A_{TL} & \star \\ \hline L_{BL}L_{TL}^T = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(a) Initial state.

$$\left(\begin{array}{c\|c} \boxed{L_{TL}} = \Gamma(A_{TL}) & \star \\ \hline L_{BL}L_{TL}^T = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(b) Top-left equation is identified as a Cholesky subproblem.

$$\left(\begin{array}{c\|c} \boxed{L_{TL}} = \Gamma(A_{TL}) & \star \\ \hline L_{BL}\boxed{L_{TL}^T} = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(c) L_{TL} becomes a known operand for the rest of equations.

$$\left(\begin{array}{c\|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL}L_{TL}^T = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(d) There is no need for algebraic manipulation.

Box 6: First iteration towards the PME generation.

In this first iteration, one unknown operand, L_{TL}, has become known, and one equation has turned into a tautology. The knowledge encoded in such a tautology is of importance for a subsequent iteration. The **second iteration** is shown in Box 7.

Structural pattern matching

Box 7(a) reproduces the final state from the previous iteration. Among the two outstanding equations, the bottom-left one is identified (Box 7(b)), as it matches the pattern of a triangular system of equations with multiple right-hand sides (TRSM). The pattern for a TRSM is

$$\{XL^T = B \ \wedge \ \texttt{Output}(X) \ \wedge \ \texttt{Input}(L) \ \wedge \ \texttt{LowerTriangular}(L) \ \wedge \ \texttt{Input}(B)\}.$$

For the sake of brevity, we assume that CLlCK had learned such pattern from a previous derivation; in practice, in case the system does not know the pattern, a nested task of PME generation would be initiated, yielding the required pattern.

Exposing new available operands

Once the TRSM is identified, the output operand L_{BL} becomes available and turns to green in the bottom-right quadrant (Box 7(c)).

Algebraic manipulation

The bottom-right equation is not in canonical form anymore: the product $L_{BL}L_{BL}^T$, now a known quantity, does not lay in the right-hand side. A simple manipulation brings the equation back to canonical form (Box 7(d)).

The process continues until all the equations are turned into tautologies. The third and **final iteration** for the Cholesky factorization is shown in Box 8, where the top formula replicates the final state from the previous iteration.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL}L_{TL}^T = A_{BL} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(a) Initial state.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline \boxed{L_{BL}} = A_{BL}L_{TL}^{-T} & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(b) Bottom-left equation is identified as a triangular system of equations.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline \boxed{L_{BL}} = A_{BL}L_{TL}^{-T} & \boxed{L_{BL}}\boxed{L_{BL}^T} + L_{BR}L_{BR}^T = A_{BR} \end{array}\right)$$

(c) L_{BL} becomes a known operand.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & L_{BR}L_{BR}^T = A_{BR} - L_{BL}L_{BL}^T \end{array}\right)$$

(d) State after the algebraic manipulation.

Box 7: Second iteration towards the PME generation.

Structural pattern matching

Only one equation, the bottom-right one, remains unprocessed. At a first glance, one might recognize a Cholesky factorization, but the corresponding pattern in Box 2 requires A to be SPD. The question is whether the expression $A_{BR} - L_{BL}L_{BL}^T$ represents an SPD matrix. In order to answer the question, CLICK applies rewrite rules and symbolic simplifications.

In Sec. 3.1.4 we explained that the following facts regarding the quadrants of A are known:

- SPD(A_{TL})
- SPD(A_{BR})
- SPD($A_{TL} - A_{BL}^T A_{BR}^{-1} A_{BL}$)
- SPD($A_{BR} - A_{BL} A_{TL}^{-1} A_{BL}^T$)

In order to determine whether $A_{BR} - L_{BL}L_{BL}^T$ is equivalent to any of the expressions listed above, CLICK makes use of the knowledge acquired throughout the previous iterations. Specifically, in the first two iterations it was discovered that

- $L_{TL}L_{TL}^T = A_{TL}$, and
- $L_{BL} = A_{BL}L_{TL}^{-T}$.

Using these tautologies as rewrite rules, the expression $A_{BR} - L_{BL}L_{BL}^T$ is manipulated. First, the equality $L_{BL} = A_{BL}L_{TL}^{-T}$ is used to replace the instances of L_{BL}, yielding $A_{BR} - A_{BL}L_{TL}^{-T}L_{TL}^{-1}A_{BL}^T$, and equivalently, $A_{BR} - A_{BL}(L_{TL}L_{TL}^T)^{-1}A_{BL}^T$. Then, by virtue of the tautology $L_{TL}L_{TL}^T = A_{TL}$, $L_{TL}L_{TL}^T$ is replaced by A_{TL}, yielding $A_{BR} - A_{BL}A_{TL}^{-1}A_{BL}^T$. Now, this expression is known to be SPD. Thanks to these manipulations, CLICK successfully associates the bottom right equation with the pattern for a Cholesky factorization.

Exposing new available operands

Once the expression in the bottom-right quadrant is identified, the system exposes the quantity L_{BR} as known. Since no equation is left, the process completes and the PME—formed by the three tautologies—is returned as output.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & L_{BR}L_{BR}^T = A_{BR} - L_{BL}L_{BL}^T \end{array}\right)$$

(a) Initial state.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & \boxed{L_{BR}} = \Gamma(A_{BR} - L_{BL}L_{BL}^T) \end{array}\right)$$

(b) Bottom-right equation is identified as a Cholesky factorization.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & \boxed{L_{BR}} = \Gamma(A_{BR} - L_{BL}L_{BL}^T) \end{array}\right)$$

(c) L_{BR} becomes a known operand.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & L_{BR} = \Gamma(A_{BR} - L_{BL}L_{BL}^T) \end{array}\right)$$

(d) Final PME.

Box 8: Final iteration towards the PME generation.

By means of the described process, PMEs for a target equation are automatically generated. The PME for the Cholesky factorization is given in Box 9.

$$\left(\begin{array}{c|c} L_{TL} = \Gamma(A_{TL}) & \star \\ \hline L_{BL} = A_{BL}L_{TL}^{-T} & L_{BR} = \Gamma(A_{BR} - L_{BL}L_{BL}^T) \end{array}\right)$$

Box 9: Partitioned Matrix Expression for the Cholesky factorization.

3.3 Non-Uniqueness of the PME

For the Cholesky factorization, CLICK identifies that only one set of partitioning rules is feasible. This corresponds to one way of decomposing the problem and to the generation of one PME. In general the PME *is not unique* as multiple sets of viable rules may be found for one target operation. In fact, each set leads to a different problem decomposition and a different PME. To illustrate such a situation, we look once more at the triangular Sylvester equation (Box 5).

The procedure described in Sec. 3.1.4 is used to obtain the sets of admissible partitioning rules, listed in Table 3. Each of these sets of rules are then applied to the postcondition equation to obtain the associated partitioned postconditions, shown in Table 4 (left). By applying the iterative process described in Sec. 3.2 to each of the partitioned postconditions, three PMEs are generated: Table 4 (right). In Box 10 we illustrate the steps performed by CLICK to transform the second partitioned postcondition into a PME.

#	Partitioned Postcondition	Partitioned Matrix Expression
1	$(L)\left(X_L \mid X_R\right) + \left(X_L \mid X_R\right)\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) = \left(C_L \mid C_R\right)$	$\left(X_L = \Omega(L, U_{TL}, C_L) \mid X_R = \Omega(L, U_{BR}, C_R - X_L U_{TR})\right)$
2	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) + \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right)(U) = \left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right)$	$\left(\begin{array}{l} X_T = \Omega(L_{TL}, U, C_T) \\ \hline X_B = \Omega(L_{BR}, U, C_B - L_{BL} X_T) \end{array}\right)$
3	$\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) + \left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right)\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) = \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$	$\left(\begin{array}{c\|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) = \left(\begin{array}{c\|c} \Omega(L_{TL}, U_{TL}, C_{TL}) & \Omega(L_{TL}, U_{BR}, C_{TR} - X_{TL} U_{TR}) \\ \hline \Omega(L_{BR}, U_{TL}, C_{BL} - L_{BL} X_{TL}) & \Omega(L_{BR}, U_{BR}, C_{BR} - L_{BL} X_{TR} - X_{BL} U_{TR}) \end{array}\right)$

Table 4: Partitioned postconditions and Partitioned Matrix Expressions for the triangular Sylvester equation.

$$\left(\begin{array}{l} L_{TL}X_T + X_T U = C_T \\ \hline L_{BL}X_T + L_{BR}X_B + X_B U = C_B \end{array}\right)$$

(a) Initial state.

$$\left(\begin{array}{l} \boxed{X_T} = \Omega(L_{TL}, U, C_T) \\ \hline L_{BL}X_T + L_{BR}X_B + X_B U = C_B \end{array}\right)$$

(b) Top equation identified as a Sylvester equation, where all the input operands are known.

$$\left(\begin{array}{l} \boxed{X_T} = \Omega(L_{TL}, U, C_T) \\ \hline L_{BL}\boxed{X_T} + L_{BR}X_B + X_B U = C_B \end{array}\right)$$

(c) X_T becomes available for the bottom equation.

$$\left(\begin{array}{l} X_T = \Omega(L_{TL}, U, C_T) \\ \hline L_{BR}X_B + X_B U = C_B - L_{BL}X_T \end{array}\right)$$

(d) State after algebraic manipulation.

$$\left(\begin{array}{l} X_T = \Omega(L_{TL}, U, C_T) \\ \hline \boxed{X_B} = \Omega(L_{BR}, U, C_B - L_{BL}X_T) \end{array}\right)$$

(e) Bottom equation is also identified as a Sylvester equation.

$$\left(\begin{array}{l} X_T = \Omega(L_{TL}, U, C_T) \\ \hline X_B = \Omega(L_{BR}, U, C_B - L_{BL}X_T) \end{array}\right)$$

(f) Resulting PME.

Box 10: PME generation for one of the possible partitionings of the Sylvester equation.

4. CONCLUSIONS

The work we presented sets the ground for the development of a symbolic system that, from the sole description of an operation, generates algorithms automatically. The core of our methodology stands in the PME. A PME encapsulates the information about the target operation in a way that facilitates the subsequent identification of loop invariants. The loop invariants then lead to the final algorithms through a technique based on program correctness. In this paper we introduce a symbolic system, CLlCK, that automates the generation of PMEs.

In order to generate PMEs, CLlCK first identifies how the operands in the operation may be partitioned. Instead of a brute force approach of exponential complexity, CLlCK utilizes a tree-based algorithm that yields only the viable sets of partitioning rules. Through a process of pattern matching, each such set leads to a distinct PME. The key in the PME generation is CLlCK's ability to identify known patterns. Initially, CLlCK only recognizes elementary structures, but its knowledge expands by automatically learning the patterns associated with the operations it tackles. Thanks to this augmenting internal knowledge, the system may generate PMEs for increasingly complex operations.

A comment on the generality of the approach follows. In order to illustrate CLlCK and the algorithms it utilizes, we discussed the Cholesky factorization and the Sylvester equation. Despite the fact that such operations differ in multiple ways—number and properties of the operands, number of valid sets of partitioning rules, number of PMEs—the steps performed by CLlCK, leading to the PMEs, are exactly the same. In a future extension, we plan to allow CLlCK to go beyond matrix equations, adding support for higher dimensional objects and the derivative operator.

5. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support received from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111.

6. REFERENCES

[1] P. Bientinesi. Mechanical derivation and systematic analysis of correct linear algebra algorithms. Technical Report TR-06-46, Department of Computer Sciences, The University of Texas at Austin, September 2006.

[2] P. Bientinesi, J. A. Gunnels, M. E. Myers, E. S. Quintana-Ortí, and R. A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31(1):1–26, Mar. 2005.

[3] P. Bientinesi, B. Gunter, and R. A. van de Geijn. Families of algorithms related to the inversion of a symmetric positive definite matrix. *ACM Trans. Math. Softw.*, 35(1):1–22, 2008.

[4] D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Springer Verlag, 1992.