Aachen Institute for Advanced Study in Computational Engineering Science

Preprint: AICES-2010/10-1

04/October/2010

Automatic Generation of Partitioned Matrix Expressions for Matrix Operations

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

Automatic Generation of Partitioned Matrix Expressions for Matrix Operations

Diego Fabregat-Traver and Paolo Bientinesi

AICES, RWTH Aachen, Germany; {fabregat,pauldj}@aices.rwth-aachen.de

Abstract. We target the automatic generation of formally correct algorithms and routines for linear algebra operations. Given the broad variety of architectures and configurations with which scientists deal, there does not exist one algorithmic variant that is suitable for all scenarios. Therefore, we aim to generate a family of algorithmic variants to attain high-performance for a broad set of scenarios.

One of the authors has previously demonstrated that automatic derivation of a family of algorithms is possible when the Partitioned Matrix Expression (PME) of the target operation is available. The PME is a recursive definition that states the relations between submatrices in the input and the output operands. In this paper we describe all the steps involved in the automatic derivation of PMEs, thus making progress towards a fully automated system.

Keywords: automation, partitioned matrix expression, algorithm generation
PACS: 02.70.Wz, 02.60.Dc

1. INTRODUCTION

Our goal is the automatic generation of a family of algorithmic variants to compute a given matrix operation. Many factors impact the performance attained by an algorithm, such as the computer architecture (shared memory, distributed memory, GPUs, etc.) and the progamming paradigm (sequential, multi threaded, blocked algorithms, algorithms-by-blocks, etc.). It is well known that there does not exist one single algorithm best suited for the whole spectrum of architectures and types of implementations; different algorithmic variants attain different performance levels on different scenarios. Therefore there is a clear need for having access to a family of algorithmic variants to compute an operation. In [1], the authors discuss in depth the need for a family of algorithmic variants.

The question is how can we obtain such a family of algorithmic variants? In [2], one of the authors presents a methodology based on formal methods and program correctness to derive a family of algorithmic variants for a target operation. Such a derivation is fully determined by the mathematical description of the operation. Therefore, the steps to be performed are mechanical and can be automatically performed by a computer.

Although the approach is mechanical, deriving algorithms by hand is tedious and error-prone. Thus, we aim to automate the derivation process by using a symbolic system. In that respect, one of the authors presented a prototype for the automatic derivation of a family of algorithmic variants [2]. Such a prototype needs as input the so-called Partitioned Matrix Expression (PME) of the operation. The PME is a matrix-like object that contains a recursive definition of the operation and represents how the different parts of the output are computed in terms of the input operands. In Fig. 1 we show as an example the PME for the *LU* decomposition.

$$\left(\begin{array}{c|c} \{L_{TL}\backslash U_{TL}\} = LU(A_{TL}) & U_{TR} = L_{TL}^{-1}A_{TR} \\ \hline L_{BL} = A_{BL}U_{TL}^{-1} & \{L_{BR}\backslash U_{BR}\} = LU(A_{BR} - L_{BL}U_{TR}) \end{array}\right)$$

where T, B, L and R are used for Top, Bottom, Left and Right, respectively.

FIGURE 1. Partitioned Matrix Expression for the *LU* decomposition.

The problem with such an approach is that the derivation of a PME is not always straightforward and can involve an important amount of algebraic manipulation and pattern matching. In this paper, we show how, given a mathematical description of a target operation, a symbolic system can generate automatically its PME. We have implemented a prototype that performs this process for a large class of matrix operations. The paper is organized as follows. In Section 2 we specify the input needed by a symbolic system. Partitionings of the operands and inheritance of properties are

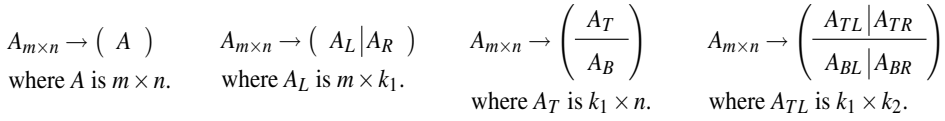

FIGURE 2. Possible partitionings for a generic matrix operand A.

discussed in Section 3 while in Section 4 we describe how to use partitionings to obtain PMEs. We draw conclusions in Section 5. We will use the *LU* factorization as a case study throughout this paper.

2. INPUT TO THE SYSTEM

The goal of this paper is to show how to automate the derivation of PMEs for a matrix operation from its formal specification. The first concern is therefore the formal specification itself. As an example, assume that we want to find the PME for the *LU* decomposition. How would such an operation be described? The most typical answer would be $LU = A$. For anyone who is familiar with the field of numerical linear algebra, this equation would be self-explanatory. The letters L and U are used to denote lower and upper triangular matrices, respectively; L is assumed to have unit diagonal; the output operands are on the left-hand side of the equation, and the input on the right-hand side.

Let us now consider the equation $XY = Z$. Associating this equation to a specific operation would be nothing more than a guess. It could represent a matrix-matrix multiplication, a solution of a system of equations, or a matrix factorization. The main reason for such a variety of interpretations is the lack of information about operands properties, mainly what is known and what is unknown in the equation.

This simple example leads to a number of observations and notational cues. First, in describing an operation we rely too much on conventions. Second, often times the notation is ambiguous. Third, the input/output conditions are often given implicitly. Since we are aiming for a fully-automated system, i.e., without any human intervention, it should now be apparent the need for a formal language to unequivocally describe the target operation. Such a language will allow us to specify all the known properties of the operands and, of course, the equation itself. The following is an example of said formalism for the *LU* decomposition:

$$f : LU = A \equiv \begin{cases} P_{pre}: & \{Unknown(L) \wedge LowerTriangular(L) \wedge UnitDiagonal(L) \wedge \\ & \ Unknown(U) \wedge UpperTriangular(U) \wedge \\ & \ A = \hat{A} \wedge ExistenceConditions(LU(A)) \; or \; \exists \, LU(A)\} \\ P_{post}: & \{LU = \hat{A} \wedge Overwrite(A, \{L, U\})\} \end{cases}$$

As we can appreciate, the formalism consists of two predicates, namely P_{pre} and P_{post}. These predicates represent the precondition (P_{pre}) and postcondition (P_{post}) of an operation. In the precondition we describe the operands that appear in the equation and their properties. The postcondition represents the equation to be solved, the target operation. Notice that this pair of predicates is the only input to our symbolic system, and the following steps are completely automatically performed.

3. PARTITIONING AND INHERITANCE

Once we have removed the ambiguities from the description of matrix operations and established a language to formally specify such operations, we are in a position to automate the process of derivation of PMEs. We recall that a PME of an operation is a recursive definition stating how parts of the output operands can be expressed in terms of parts of the input operands. Thus, the first step is to partition the operands of the operation. In Fig. 2 we list the possible partitionings for a generic matrix operand A. The subscript letters T, B, L, and R are used for *T*op, *B*ottom, *L*eft, and *R*ight, respectively. Note that, so far, we have not made any assumptions on the size of the partitions.

When partitioning operands, it is often beneficial to preserve their structure and properties. For instance, if a matrix is lower triangular, it is convenient to partition it in such a way that some of the resulting partitions inherit said property and thus being able to exploit it in subsequent steps of the derivation process. A 1×2 or a 2×1 partitioning breaks the structure, and no partitions inherit the lower triangularity. Even a 2×2 partitioning is not enough to guarantee the

$$L \rightarrow \left(\begin{array}{c} L \end{array} \right) \qquad\qquad L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right)$$

a) 1×1 partitioning. L is lower triangular. b) 2×2 partitioning. L_{TL} and L_{BR} are lower triangular.

FIGURE 3. Example of possible partitionings for lower triangular matrices.

$$L \rightarrow \left(\begin{array}{c} L \end{array} \right); U \rightarrow \left(\begin{array}{c} U \end{array} \right); A \rightarrow \left(\begin{array}{c} A \end{array} \right) \qquad L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right); U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right); A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$$

a) All operands partitioned 1×1. b) All operands partitioned 2×2.

FIGURE 4. Possible partitionings for the *LU* decomposition.

triangularity of the TL and BR quadrants so we need to enforce that TL is square. We illustrate this with an example of property inheritance in Fig. 3.

Another constraint to be satisfied by the partitioning of the operands is that its combination has to be conformal with respect to addition and multiplication. Therefore, not every combination of partitionings is feasible.

To sum up these ideas, there are basically four ways of partitioning an object, as seen in Fig. 2. However, depending on the structure of the operands, some restrictions may apply to their partitioning so that some of them are not adequate (Fig. 3). After partitioning the operands, some of the submatrices might inherit a structure that an automated system will be able to exploit in the next steps of the derivation. Also, the partitioning of the operands must be conformal, respecting the restrictions enforced by the equation to be solved. One extra complication is that more than one combination of partitionings is possible in some operations, so an automated system has to be able to generate all of them.

We show a final example of the partitioning process in Figs. 4 and 5. In Fig. 4, we show the possible partitionings of the operands for an *LU* decomposition. As we can appreciate, there are two sets of feasible partitionings for such operation. However, we will discard the one shown in Fig. 4a because no partitioning is made in any of the operands and, therefore, it does not lead us to a loop-based algorithm. Finally, in Fig. 5, we show what we call the partitioned postcondition, which is the equation stated in the *postcondition* predicate where the operands have been replaced by their *partitioned* counterparts. Looking at Fig. 5 and comparing it with Fig. 1, we can get the feeling of where we are going and maybe anticipate the next steps towards the PME derivation.

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) \cdot \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$$

FIGURE 5. Partitioned postcondition for the *LU* decomposition. Here, L_{TL}, U_{TL} and A_{TL} are square.

4. GENERATION OF THE PME

In order to describe how to obtain a PME from a partitioned postcondition, we introduce the canonical form for a PME. PMEs are matrix-shaped objects where each of the elements are equations. We impose that each equation has the output operands on the left-hand side and the input operands on the right-hand side; i.e., we impose an explicit definition of the equations. An example of this structure is shown in Fig. 1. The PME illustrates the dependencies among the equations.

The generation of PMEs from partitioned postconditions consists of two phases. First, the symbolic system automatically performs symbolic simplifications to merge the partitioned postcondition into a single matricial expression; i.e., it adds and multiplies out the blocked matrices. In the case of the *LU* decomposition, performing this step on the expression in Fig. 5 results in the following expression:

$$\left(\begin{array}{c|c} L_{TL} \cdot U_{TL} = A_{TL} & L_{TL} \cdot U_{TR} = A_{TR} \\ \hline L_{BL} \cdot U_{TL} = A_{BL} & L_{BL} \cdot U_{TR} + L_{BR} \cdot U_{BR} = A_{BR} \end{array} \right)$$

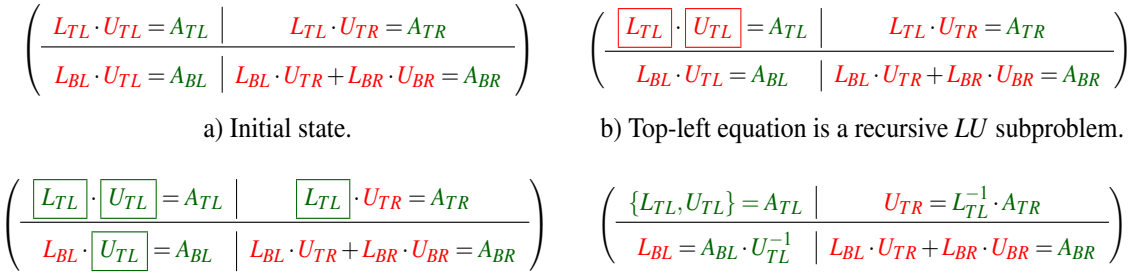

FIGURE 6. First iteration towards the canonical form of the PME.

In the second phase, the symbolic system automatically performs steps of algebraic manipulation and pattern matching to identify relations between submatrices of the input and the output operands. During this iterative process three actions are carried out: 1) structural pattern matching, 2) input/output classification, and 3) algebraic manipulation.

To illustrate the second stage we show in Fig. 6 the steps for the first iteration in the LU decomposition. In the figure, green and red are used to highlight the input and output operands, respectively. Fig. 6a shows the initial state where the operands on the left-hand side of the equations are outputs and the operands in the right-hand side are inputs. The only equation that can be identified is the top-left one, an LU decomposition (Fig. 6b). As a result, L_{TL} and U_{TL} become input operands for the rest of equations (Fig. 6c). Finally, an algebraic manipulation step is performed, where the top-right equation is left multiplied by L_{TL}^{-1} and the bottom-left equation is right multiplied by U_{TL}^{-1}, obtaining Fig. 6d.

The symbolic system continues with similar steps: the top-right and bottom-left equations represent two solutions of triangular systems, and bottom-right equations corresponds to a recursive LU decomposition. The result of the process is the PME as shown in Fig. 1. This expression is a recursive definition of the target operation, it exposes the operations that need to be performed to carry out the overall LU factorization of the input matrix A. It also contains the dependencies between these operations, therefore exposing a necessary ordering for them.

PMEs are the key objects in our methodology for automating the generation of algorithms. All the information included in the PMEs is used to derive a family of loop invariants, which guide the final steps of algorithms generation. For a description of the loop invariants and the generation of algorithms from a PME and its loop invariants, we refer the reader to [2].

5. CONCLUSIONS

We exposed the input needed by a symbolic system to derive algorithms and the necessary steps to automatically generate the Partitioned Matrix Expression for a matrix operation. Along these steps, we identified the ambiguity of the customary notation, enumerated the restrictions to be taken into account when partitioning the operands, and illustrated how to identify the dependencies between the operations involved in the algorithm.

In combination with a previous prototype that automatically generates algorithms from a PME, the system described in this paper represents a concrete improvement towards a fully automated code generator.

ACKNOWLEDGMENTS

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

REFERENCES

1. Bientinesi, P., Gunter, B. and Van de Geijn, Robert A.: Families of Algorithms Related to the Inversion of a Symmetric Positive Definite Matrix. ACM Transactions on Mathematical Software, 35(1), July 2008.
2. Bientinesi P.: Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms. Ph.D. Dissertation. Department of Computer Sciences, The University of Texas. Technical Report TR-06-46. September 2006.