CHAPTER 78

A Linear Algebraic View of Loop Transformations and Their Interaction

Jagannathan Ramanujam*

Abstract. Although optimizing transformations have been studied for over two decades, the interactions between them is not well understood. This is particularly important for the success of parallelizing compilers. In order to deal with interactions, we view loop transformations as multiplication by a suitable matrix. The transformations considered are loop interchange, permutation, reversal, hyperplane (skewing), tiling and concurrentization. Since these transformations are realized as matrix products, this is easy to implement in a compiler. We show an example where loop interchange by itself is illegal but becomes legal when considered after loop reversal. We show that unimodular transformations are not sufficient for minimizing communication volume arising from inter-tile communication in the case of tiling. We present several additional examples of the use of this approach.

Key words. Parallelizing compilers, loop transformations, linear algebraic view of transformations, interaction among transformations, program optimization

1. Introduction. Although optimizing transformations have been studied for over two decades resulting in a substantial catalog of transformations, the interactions between them is not well understood. This problem is of particular importance for the success of parallelizing compilers. The order of applying a set of transformations has a great impact on the performance of the loop. The problem is exacerbated by the underlying machine architecture. In addition, it is becoming increasingly common for programmers of highly concurrent machines to specify parallelism using language constructs such as DOALL. Hence, there is a need for a framework to deal with interactions between optimizations.

This paper presents an approach towards this problem. We present a linear algebraic view of a subset of loop transformations that results in a single notation for expressing those transformations. Many loop transformations are linear transformations applied to the iteration space and the dependence matrix. We use the matrix associated with the linear transformation. Applying transformations $T_1, T_2, \ldots, T_{k-1}, T_k$ (in that order) is equivalent to the matrix product $M_k \times M_{k-1} \times \ldots \times M_2 \times M_1$ where M_i is the matrix associated with the transformation T_i. With iteration space tiling, this transformation

* Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803-5901. jxr@max.ee.lsu.edu

is a measure of inter-tile communication volume. In addition to these advantages, the approach provides a controlled search of the space of possible loop optimizations. We show several examples of how the technique presented here deals with interactions among transformations.

2. Background and related work.

2.1. Related work. Dowling [7] presents an integer programming solution to the problem of optimal code parallelization using unimodular matrices transform a given nested loop into a parallel form. Banerjee [6] and Wolf and Lam [12] present unimodular transformations of loop nests and algorithms to derive transformations for maximal parallelism.

A recent paper by Whitfield and Soffa [11] present an approach to dealing with interactions among transformations based on a Hoare-like axiomatic specification technique. Wolfe [17] presents a technique that searches through a search space comprising both linear and non-linear transformations of the iteration space.

2.2. Background. The transformations considered are loop interchange [1, 2, 3, 14, 15] permutation [5], reversal, hyperplane (skewing) [9, 13], tiling [8, 10, 16] and concurrentization. The presentation here deals with dependence distance vectors; dependence analysis is dealt with in detail in [1, 3, 4, 15, 18].

We assume familiarity with the notions of dependence. A dependence matrix a collection of dependence distance vectors for a nested loop – the distance vectors are the columns of the dependence matrix. For a d-nested loop with n dependence vectors $d_i, i = 1, \ldots, n$, the dependence matrix D is a $d \times n$ matrix. Let $i_1, i_2, \ldots, i_d$ be the loop index variables for the loop nest levels. The vector $(i_1, i_2, \ldots, i_d)$ is referred to as the *nest vector*.

3. Loop transformations. Loop transformations are applied to the loop nest vector and the dependence matrix. Let $\vec{I} = (I_1, \ldots, I_d)$ and $\vec{J} = (J_1, \ldots, J_d)$ be two iterations of a nested loop such that $\vec{I}$ precedes $\vec{J}$ and there is a dependence of constant distance $\vec{d_i}$ between them. Applying a linear transformation T to the iteration space (nest vector) also changes the dependence matrix since $T(\vec{J}) - T(\vec{I}) = T(\vec{J} - \vec{I}) = T\ \vec{d_i}$. A transformation is legal if the resulting dependence vectors are *positive* vectors [4]. Thus, the linear algebraic view leads to a simpler notion of the legality of a transformation. For a d-nested sequential loop, the $d \times d$ identity matrix (I_d) denotes sequential execution order. We deal with perfect nests.

Loop interchange and permutations. Loop interchange (for doubly nested loops) is viewed as the product of the following matrix (which is the transpose of the 2×2 identity matrix)

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and the nest vector (i_1, i_2) where i_1 and i_2 are the loop indices. Since the common usage for such a nest is (i, j) we will use that without loss of generality. For three level nest we will use (i, j, k) for the same reason. If interchange is valid, the new nest vector is:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} j \\ i \end{bmatrix}$$

Loop interchange (which is given by a matrix T) will be valid if the first non-zero entry in each column of the transformed dependence matrix $D^{'}$ is positive where

$$D^{'} = T\ D$$

for $i = 2$ to N
  for $j = 1$ to $N - 1$
    $A[i,j] \leftarrow A[i-1,j] + A[i-1,j+1]$
  endfor
endfor

FIG. 1. **Example 1**

where D is a $d \times n$ dependence matrix. We point out Allen and Kennedy [2] mention that interchange can be thought of multiplying by the transpose matrix.

Loop permutations are given by a class of $d \times d$ assignment matrices (where each row contains exactly one 1 and each column contains exactly one 1 and all other entries are 0). For a d-level nest there are $d!$ assignment matrices. It had been suggested that loop permutations can be viewed as a sequence of loop interchanges [2]; Banerjee [5] points out that a given loop permutation may be realized as one of possibly many sequences of loop interchanges, some of which could be invalid. The notation used here can account for that easily. Let a loop permutation P be realized as a sequence of k loop interchanges $T_1, \ldots, T_k$. Therefore,

$$P = T_1 \times T_2 \times \cdots T_k$$

where each of the matrices $P, T_1, T_2, \ldots, T_k$ are $d \times d$ permutation matrices and each matrix T_i differs from T_{i-1} in exactly two columns (or rows). It is possible that

$$\left(\prod_{i=1}^{m} T_i \right) D \qquad m < k$$

results in a transformed matrix in which the first non-zero entry in some column is negative. Loop circulation is a special case of permutations and corresponds to d out of $d!$ permutation matrices for d-nested loops.

Loop reversal. Loop reversal is running the loop backwards and is legal if the loop carries no dependence. Reversal is accomplished by a transformation T where in order to reverse the nest level k, the entry $T_{k,k}$ is -1. The sequential execution order for normalized loops of nest d is given by the $d \times d$ identity matrix. Since loop reversal is specified by a variant of assignment matrices where each row (or column) contains exactly one non-zero entry and the non-zero entries are limited to $+1$ and -1, it can always be thought of in conjunction with loop permutations. In the loop example in Figure 1, loop interchange can not be applied because of the $(1, -1)$ dependence. But if the j loop is reversed, interchange can be applied. The transformation applied is:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

A combination of permutations and reversals is generated by a class of $d \times d$ matrices whose entries belong to the set $\{0, 1, -1\}$ and there is exactly one nonzero entry in each row or column of such matrices.

Loop skewing – Hyperplane Transformation. The hyperplane transformation or loop skewing can be viewed as a transformation realized through a lower triangular matrix T with unit diagonal. Since the matrix is lower triangular with non-zero diagonal, it is non-singular. Consider example 2 shown in Figure 2. Skewing the j loop forward by

for $i = 2$ to N
  for $j = 2$ to N
    $A[i,j] \leftarrow A[i-1,j] + A[i,j-1]$
  endfor
endfor

FIG. 2. **Example 2**

for $i = 2$ to N
  for $j = 2 + i$ to $N + i$
    $A[i,j-i] \leftarrow A[i-1,j-i] + A[i,j-i-1]$
  endfor
endfor

FIG. 3. *Example 2 after skewing*

a factor of 1 with respect to the i loop will result in the loop shown in Figure 3.

The loop skew accomplished is realized through the matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Skewing the loop i_k by a factor f with respect to the loop $i_l (l < k)$ in a nest $(i_1, \ldots, i_l, \ldots, i_k, \ldots, i_d)$ is accomplished by matrix T whose kth row differs from the kth row of I_d in that *row* $k \leftarrow$ *row* $k + f \times$ *row* l.

3.1. Elementary row operations. All the transformations discussed above correspond to elementary row transformations of matrices [4]:

- Interchanging rows corresponds to loop interchange, permutation and circulation.
- Multiplying a row by -1 corresponds to loop reversal.
- Replacing a given row k by row k plus and integral multiple of another row i is equivalent to loop skewing.

Thus, any unimodular transformation can be realized through reversal, interchange and skewing.

In the case of multiply nested loops, the order of applying loop skewing can affect the final result – in a nest given by (i, j, k) skewing j loop with respect to i and then skewing k loop with respect to j ("outside-in" strategy) results in a different final loop from the "inside-out" strategy (skewing k loop with respect to j first and then skewing j with respect to i loop). Viewing this sequence of skews as a sequence of matrix multiplications easily accounts for the different final results. Thus the view presented here accounts for *composability* of transformations.

4. Tiling and Nonunimodular Matrices. Loop tiling [8, 10, 15, 16] is a transformation that decomposes a d-dimensional iteration space into d-dimensional "blocks" composed of a number of iterations where each block executes atomically; hence there should be no dependence cycles (deadlock) among the blocks. Tiling is extremely useful in distributed memory machines and hierarchical shared memory machines in exploiting locality of reference in the presence of dependences.

Each tile boundary is defined by a vector normal to it. To be legal, there must be no dependence cycles among tiles. In d dimensions, any d vectors $H_1, \ldots, H_d$ (where H_i is the vector normal to the ith boundary) define legal tiles if

$$H_i \cdot \vec{d_j} \geq \vec{0}, \qquad i = 1, \ldots, d \tag{1}$$

for all d_j in the set of dependence vectors. Each of the $d-1$ dimensional planes is a boundary whose normal is one of the H_i. To form d-dimensional tiles from a d-dimensional iteration space, the vectors normal to the d tile boundaries should be linearly independent. Let H denote the $d \times d$ matrix whose rows are the vectors $H_1, \ldots, H_d$. In matrix notation, let $D^+ = HD$. The tiling condition implies that, for tiles to be legal

$$D^+_{ij} \geq 0 \qquad i = 1, \ldots, d \qquad j = 1, \ldots, n \tag{2}$$

Thus all entries in D^+ must non-negative. Therefore, *all loop permutations are legal* which is a necessary condition for generating strip-mined code with all the strip loops moved to the outer level. Finding the tiling planes can be formulated as (a linear programming problem) that of finding a transformation H such that

$$\sum_{k=1}^{d} H_{i,k} D_{k,j} \geq 0 \quad i = 1, \ldots, d \quad j = 1, \ldots, n$$

and H must be nonsingular.

4.1. Tiling for minimal communication. For a unimodular matrix H defining tile boundaries as discussed above, a non-zero entry in D^+ say D^+_{ij} implies that communication is incurred due the jth dependence vector poking the ith tile boundary. The amount communication across a tile boundary defined by the ith row of H is a function of the sum of the entries in the ith row of the transformed dependence matrix D^+. The communication volume for an $l \times l \times \cdots \times l$ tile is:

$$\text{CommunicationVolume} = l^{d-1} \sum_{i=1}^{d} \sum_{j=1}^{n} \sum_{k=1}^{d} H_{i,k} D_{k,j} \tag{3}$$

Thus, the problem of finding a set of valid tiling planes that minimizes communication volume is equivalent to minimizing $\sum_{i=1}^{d} \sum_{j=1}^{n} D^+_{i,j}$.

4.2. Nonunimodularity and Communication. In the case of 2-dimensional iteration spaces, the transformation matrix is not necessarily unimodular [10]. For example, consider the dependence matrix

$$D = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & -1 & 1 & 1 \end{bmatrix}.$$

Among unimodular matrices of the form

$$H = \begin{bmatrix} 1 & 0 \\ a & 1 \end{bmatrix},$$

$a = 1$ gives the minimum communication volume whose value is 11. If we look for nonsingular matrices of the form

$$H = \begin{bmatrix} 1 & h_1 \\ h_2 & 1 \end{bmatrix},$$

$h_1 = -1$ and $h_2 = 1$ gives the minimum value of $\sum_{i=1}^{d} \sum_{j=1}^{n} D^+_{i,j}$ which is 10. We note that H in this case has a determinant 2 and thus can not be generated by unimodular transformations. The reader is referred to [10] for details.

5. Discussion. In order to deal with *interactions* among loop transformations and to address the problem of *composing* them, we view loop transformations as multiplication by a suitable matrix. The transformations considered are loop interchange, permutation, reversal, hyperplane (skewing), tiling and concurrentization. Since these transformations are realized as matrix products, this is easy to implement in a compiler. We showed an example where loop interchange by itself is illegal but becomes legal when considered after loop reversal. We also showed that unimodular matrices are not sufficient for minimizing communication volume in tiling.

REFERENCES

[1] J. R. Allen. *Dependence Analysis for Subscripted Variables and its Applications to Program Transformations.* Ph.D. Dissertation, Department of Mathematical Sciences, Rice University (UMI 83-14916), Houston, Texas, April 1983.

[2] J. R. Allen and K. Kennedy. Automatic Loop Interchange. *Proc. SIGPLAN 84 Symposium on Compiler Construction,* pages 233–246, June 1984.

[3] J. R. Allen and K. Kennedy. Automatic Translation of FORTRAN Programs to Vector Form. *ACM Trans. Programming Languages and Systems,* Vol. 9, No. 4, pages 491–542, October 1987.

[4] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Boston, Massachusetts, 1988.

[5] U. Banerjee. A Theory of Loop Permutations. *Languages and Compilers for Parallel Computing,* D. Gelernter, A. Nicolau and D. Padua (Eds.) Pitman, London, 1990, pp. 54–74.

[6] U. Banerjee. Unimodular Transformation of Double Loops. *3rd Workshop on Languages and Compilers for Parallel Computing,* Aug. 1990.

[7] M. Dowling. Optimal Code Parallelisation using Unimodular Transformations. *Preprints in Optimization, Carolo-Wilhelmina Universität zu Braunschweig, W. Germany,* July 1988.

[8] F. Irigoin and R. Triolet. Supernode Partitioning. *Proc. 15th Annual ACM Symp. Principles of Programming Languages,* San Diego, CA, Jan. 1988, 319-329.

[9] L. Lamport. The Parallel Execution of DO Loops. *Communications of the ACM,* Vol. 17, No. 2, pages 83–93, February 1974.

[10] J. Ramanujam. *Compile-time Techniques for Parallel Execution of Loops on Distributed Memory Multiprocessors.* Ph.D. Thesis, The Ohio State University, September 1990.

[11] D. Whitfield and M. Soffa. An Approach to Ordering Optimizing Transformations. *Proc. Second PPoPP,* SIGPLAN Notices, Vol. 25, No. 3, pages 137–146, March 1990.

[12] M. Wolf and M. Lam. Maximizing Parallelism via Loop Transformations. *3rd Workshop on Languages and Compilers for Parallel Computing,* Aug. 1990.

[13] M. Wolfe. Loop Skewing: The Wavefront Method Revisited. *International Journal of Parallel Programming,* Vol. 15, No. 4, pages 279–294, 1986.

[14] M. Wolfe. Advanced Loop Interchange. *Proc. 1986 International Conference on Parallel Processing,* St. Charles, IL, pages 536–543, August 1986.

[15] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* Pitman Publishing, London and the MIT Press, Cambridge, Massachusetts, 1989.

[16] M. Wolfe. More Iteration Space Tiling. *Proc. Supercomputing 89,* Reno, Nevada, pages 655–664, November 1989.

[17] M. Wolfe. Massive Parallelism through Program Restructuring. *Tech. Report CS/E 90-009,* Oregon Graduate Institute, June 1990.

[18] M. Wolfe and U. Banerjee. Data Dependence and Its Application to Parallel Processing. *International Journal of Parallel Programming,* Vol. 16, No. 2, pages 137–178, 1987.