To appear in Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, February 1995.

Integrating Data Distribution and Loop Transformations*

J. Ramanujam† A. Narayan†

Abstract

This paper presents a technique for finding good distributions of arrays and suitable loop restructuring transformations so that communication is minimized in the execution of nested loops on message passing machines. For each possible distribution (by one or more dimensions), we derive the best unimodular loop transformation that results in block transfers of data. Unlike other work which focus on either data layout or on program transformations, this paper combines both array distributions and loop transformations resulting in good performance. The techniques described here are suitable for dense linear algebra codes.

1 Introduction

On a distributed memory machine, local memory accesses are much faster than accesses to non-local data. Inter-processor communication—resulting accesses to non-local data—is a major determinant of the performance of a parallel machine. When a number of non-local accesses are to be made between processors, it is preferable to send fewer but larger messages rather than several smaller messages more frequently (called *message vectorization*). This is because the message setup cost is usually large. Even in shared memory machines, it is preferable to use block transfers.

Given a program segment, our aim is to determine the computation and data mapping onto processors. Parallelism can be exploited by transforming the loop nest suitably and then distributing the iterations of the transformed outermost loop onto the processors. The distribution of data onto processors may then result in communication and synchronization which counters the advantages obtained by parallelism. This paper presents an algorithm which results in the optimal performance while simultaneously considering the conflicting goals of parallelism and data locality.

While a programmer can manually write code to enhance data locality by specifying data distribution among processors, we present a technique where we can automatically derive data distribution given the program structure. We present a method by which the program is restructured such that when the outer loop iterations are mapped onto the processors, it results in the least communication. Wherever communication is unavoidable, we restructure the inner loop(s) so that data can be transferred using block transfers; such an approach is referred to as *message vectorization.*

In this paper, we consider the cases where we allocate outer iterations to processors so that each outer loop iteration is done by a single processor. The data is then allocated so that there is minimum communication and all communication is done through block transfers. This paper deals with an algorithm to restructure the program to enhance data locality while still enabling parallelism. We construct the entries of a legal invertible transformation matrix so that there is a one-to-one mapping from the original iteration space to the transformed iteration space. This transformation when applied to the original loop structure will do the following:

- Allow the outermost most loop iteration to be distributed over the processors *i.e.,* an entire outermost iteration is mapped on to a single processor.
- Determine the data distribution (block or cyclic distribution of a single array dimension).
- Allow blocks transfers to be moved out of the innermost loop so that all the necessary data are transferred to the respective local memories before the execution of the innermost loop.

*The first author is supported in part by an NSF Young Investigator Award (CCR–9457768), NSF grant CCR–9210422, and by the Louisiana Board of Regents through contract LEQSF (1991-94)-RD-A-09.

†Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803-5901 (jxr@gate.ee.lsu.edu).

2 Background and Terminology

The transformation matrix is derived from the *data reference matrix* of the array references. Given a loop nest with indices $i_1, i_2, \cdots, i_n$ which is represented by a column vector $\vec{I}$, we define a *data reference matrix*, $A_{\mathcal{R}}$, for each array reference A (distinct or non-distinct) in a loop nest such that the array reference can be written in the form $A_{\mathcal{R}}\vec{I} + \vec{b}$ where $\vec{b}$ is the offset vector.

Example 1:
for $i = 1$ to N_1 do
  for $j = 1$ to N_2 do
    for $k = 1$ to N_3 do
      $B[i, j-i] = B[i, j-i] + A[i, j+k]$

In the above example, the data reference matrix for the array B is $B_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix}$, and the data reference matrix for array A is $A_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. Note that there are two data reference matrices for the array B though they are identical. For each array, we use only the distinct data reference matrices.

2.1 Effect of a Transformation

On applying a transformation T to a loop with index I, the transformed loop index becomes $\vec{I'} = T\vec{I}$ and the transformed data reference matrix becomes $A'_{\mathcal{R}} = A_{\mathcal{R}}T^{-1}$. The columns of T^{-1} determine the array subscripts of the references in the transformed loop. The key aspect of the algorithm presented in this paper is that the entries of the inverse of the transformation matrix are derived using the data reference matrices.

2.2 Related work

Li and Pingali [7] discuss the completion of partial transformations derived from the data access matrix of a loop nest; the rows of the data access matrix are subscript functions for various array accesses (excluding constant offsets). Their work assumes that all arrays are distributed by columns. In contrast, our work attempts to find the best distribution for various arrays (by rows, columns, or blocks) such that communication incurred is minimal; for each possible combination of distribution of arrays, we find the best compound loop transformation that results in least communication. Among all these possible distributions (and the associated loop restructuring), we find the one that incurs the smallest communication overhead. Several researchers have addressed the issue of automatic alignment [2, 3, 4, 5, 6, 8, 10]. None of these except [1] addresses the interaction of program transformations and data mapping.

2.3 Motivation

Consider Example 1 above which is similar to the one in [7]. There are two references to the array B (though not distinct) and one reference to the array A. Li and Pingali [7] assume that all arrays are distributed by columns and derive a transformation matrix that matches column-distribution. Unlike in [7] the loop can be distributed in such a way that there is no communication incurred. Both the arrays can be distributed by rows, *i.e.,* each processor can be assigned an entire row of array A and an entire row of array B. This makes the loop run without any communication. We notice that the first row in the data reference matrix for the arrays A and B are the same *i.e.,* $\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$. This allows the first dimension of both the arrays to be distributed (*i.e.,* by rows) over the processors so that there is no communication. In the next section, we derive an algorithm to construct a transformation matrix, which determines the distribution of data.

3 Algorithm

We restrict our analysis to affine array references in loop nests whose upper and lower bounds are affine. We assume that the iterations of the outermost loop are distributed among processors. To exploit data locality and reduce communication among processors, we further look at transformations that facilitate block transfers so that the data elements which are referenced are brought to local memory in large chunks; this allows to amortize the high message start-up costs over large messages. We assume that the data can be distributed along any one dimension of the array (wrapped or blocked). The results can be easily generalized where data is distributed along multiple dimensions and block transfers set up in outer iterations.

3.1 Criteria for Choosing the Entries in the Transformation Matrix

Let the array indices of the original loop be $i_1, i_2, \ldots, i_n$. Let the array indices of the transformed loop be $j_1, j_2, \ldots, j_n$. We look for transformations such that the LHS array has the outermost loop index as the only element in any one of the dimensions of the array, *e.g.* $C(*, *, \ldots, j_1, \ldots, *)$ where j_1 is in the r^{th} dimension and "$*$" indicates a term independent of j_1. The LHS array can then be distributed along dimension r. This means that the data reference matrix of the transformed array reference C, *i.e.,* $C'_{\mathcal{R}}$ has at least one row which has the first entry as non-zero and the rest as zero. For all arrays that appear on the right hand side:

- If a row in all the data reference matrices of an array is identical to a row in the reference matrix in the LHS array, then that array can be distributed in the same way as the LHS array. There is no communication due to that array, since they are always mapped onto the same processor. If all the references of all the arrays have a row in the data reference matrix identical to that of the LHS array, then the entire loop can be distributed along that dimension and there is no communication.

- If the above condition does not hold, choose the entries in T^{-1} such that one dimension of the RHS has only the innermost loop index, *e.g.* $A(*, \ldots, j_n, \ldots, *)$ and all the other dimensions are independent of the innermost loop index (that is, "$*$" indicates a term independent of j_n). This allows a block transfer to the local memory before the execution of the innermost loop. This means that a row in the transformed data reference matrix $A'_{\mathcal{R}}$ has a row with all entries zero except in the last column, which is non-zero. Also, the last column of the $A'_{\mathcal{R}}$ has all remaining entries as zero.

The transformation should also satisfy the condition that the determinant is non-zero and must preserve the dependences in the program.

3.2 The Algorithm

Consider the following loop where n is the loop nesting level and d the dimension of the arrays.

for $l_1 = 1$ to N_1 do
 ...
 for $l_n = 1$ to N_n do

$$L\left[\begin{pmatrix} c_{11} & \ldots & c_{1n} \\ & \ddots & \\ c_{d1} & \ldots & c_{dn} \end{pmatrix}\begin{pmatrix} i_1 \\ \vdots \\ i_n \end{pmatrix} + \begin{pmatrix} b^l_1 \\ \vdots \\ b^l_d \end{pmatrix}\right] = R\left[\begin{pmatrix} a_{11} & \ldots & a_{1n} \\ & \ddots & \\ a_{d1} & \ldots & a_{dn} \end{pmatrix}\begin{pmatrix} i_1 \\ \vdots \\ i_n \end{pmatrix} + \begin{pmatrix} b^r_1 \\ \vdots \\ b^r_d \end{pmatrix}\right]$$

Let the inverse of the transformation matrix be $T^{-1} = \begin{pmatrix} t_{11} & \ldots & t_{1n} \\ & \ddots & \\ t_{n1} & \ldots & t_{nn} \end{pmatrix}$. Let $\vec{c_i}$ be the i^{th} row of the reference matrix of the LHS array and $\vec{a^p_i}$ be the i^{th} row of the reference matrix of the p^{th} *distinct* RHS array. Let $\vec{t_j}$ be the j^{th} column of T^{-1}. The algorithm is shown in Figure 1.

4 Examples

We illustrate the use of the algorithm through several examples in this section. The reader is referred to [9] for a detailed discussion of the algorithm.

Step 0: If a row in the reference matrix of all the arrays are the same, then there is no communication involved. The data can be distributed along the respective dimension and all the data for computation will be in local memory. (Initialize $i \leftarrow 1$).

Step 1: Distribute LHS array along dimension i, *i.e.*, set: $\vec{c_i}.[T^{-1}] = (1\ 0\ \cdots\ 0)$.

Step 2: Choose an RHS array which does not have a row in the reference matrix the same as that of LHS array. For all j set: $\vec{a^p_j}.[T^{-1}] = (0\ 0\ \cdots\ 0\ 1)$ for all references to that array and $\overline{a_{k \neq j}} \cdot \vec{t_n} = 0$. If a valid T^{-1} is found, check the determinant of T^{-1}. If non-zero block transfers are possible for that RHS array, (break) go to Step 3. If there are no valid T^{-1} or the determinant of T^{-1} is zero for all j, block transfers are not possible on that array with the given distribution of the LHS array; therefore, increment j and go to Step 2.

Step 3: Repeat Step 2 for all the reference matrices of a particular array.

Step 4: Repeat Step 2 for all distinct arrays on RHS. (Increment p)

Step 5: Check the number of arrays where block transfers are possible.

Step 6: Repeat Step 1 to Step 4 for LHS array distributed along all dimensions (Increment i).

Step 7: Compare the number of arrays that can have block transfers and distribute LHS array along the dimension which yields maximum number of block transfers for the arrays on the right hand side.

FIG. 1. *Algorithm for data distribution and loop transformations*

Example 2: Matrix Multiplication

```
for i = 1 to N do
  for j = 1 to N do
    for k = 1 to N do
      C(i, j) = C(i, j) + A(i, k) * B(k, j)
```

The reference matrices of the arrays are: $C_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $A_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, and $B_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

Step 1: (C distributed along first dimension). Set $\vec{c_1} \cdot \vec{t_1} = 1$, $\vec{c_1} \cdot \vec{t_2} = 0$, and $\vec{c_1} \cdot \vec{t_3} = 0$. Therefore we have, $t_{11} = 1$, $t_{12} = 0$, and $t_{13} = 0$.

Step 1a: (Derive distribution of Array A) Since row 1 of A is the same as that of C, distribute A and C identically.

Step 2.1: (Derive distribution for Array B) Set $\vec{b_1} \cdot \vec{t_1} = 0$, $\vec{b_1} \cdot \vec{t_2} = 0$, and $\vec{b_1} \cdot \vec{t_3} = 1$. Therefore we have, $t_{31} = 0$, $t_{32} = 0$, and $t_{33} = 1$; and $\vec{b_2} \cdot \vec{t_3} = 0 \Longrightarrow t_{23} = 0$. Finally we have, $T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ t_{21} & t_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$. For a unimodular transformation, $t_{22} = \pm 1$. Note that dependence vector is $(0\ 0\ 1)$, and therefore, there are no constraints on t_{21}. This results in the identity matrix as the transformation matrix, and thus nothing need be done. Distribute A and C by rows, and B by columns. The shown next gives the best performance we can get in terms of parallelism and locality.

```
for u = 1 to N do
  for v = 1 to N do
    read B(*, v)
```

for $w = 1$ to N
  $C(u, v) = C(u, v) + A(u, w) * B(w, v)$

We go ahead and complete the algorithm by looking at distributing the LHS array in the next dimension.

Step 1.1: (C distributed along second dimension). Set $\vec{c_2} \cdot \vec{t_1} = 1$, $\vec{c_2} \cdot \vec{t_2} = 0$, and $\vec{c_2} \cdot \vec{t_3} = 0$. Therefore we have, $t_{21} = 1$, $t_{22} = 0$, and $t_{23} = 0$.

Step 1.1a: (Derive distribution for Array B). Since B has row 2 same as that of C distribute B same as C.

Step 2.2: (Derive distribution for Array A). Set $\vec{a_1} \cdot \vec{t_1} = 0$, $\vec{a_1} \cdot \vec{t_2} = 0$, and $\vec{a_1} \cdot \vec{t_3} = 1$. Therefore we have, $t_{11} = 0$, $t_{12} = 0$ and $t_{13} = 1$; and $\vec{a_2} \cdot \vec{t_3} = 0 \Longrightarrow t_{33} = 0$.

Finally we have, $T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ t_{31} & t_{32} & 0 \end{pmatrix}$. For a unimodular transformation, $t_{32} = \pm 1$. Therefore, $T^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ and $T = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. Distribute the arrays A, B, and C by columns. The transformed loop is given below:

for $u = 1$ to N do
  for $v = 1$ to N do
    read $A(*, v)$
    for $w = 1$ to N do
      $C(w, u) = C(w, u) + A(w, v) * B(v, u)$

We see that the performance of the loop is similar in both the cases. Therefore the array C can either be distributed by columns with the above transformation, or by rows with no transformation for the same performance with respect to communication. Consider the SYR2K (from BLAS) example shown below.

Example 3: SYR2K
for $i = 1$ to N do
  for $j = i$ to $min(i + 2b - 2, N)$ do
    for $k = max(i - b + 1, j - b + 1, 1)$ to $min(i + b - 1, j + b - 1, N)$ do
      $C[i, j - i + 1]$ += $A[k, i - k + b] * B[k, j - k + b] + A[k, j - k + b] * B[k, i - k + b]$

The reference matrices for the arrays are: $C_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix}$, $A^1_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \end{pmatrix}$, and $A^2_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}$. In addition, $B^1_{\mathcal{R}} = A^2_{\mathcal{R}}$ and $B^2_{\mathcal{R}} = A^1_{\mathcal{R}}$.

Step 1: (C row distributed) Set $\vec{c_1} \cdot \vec{t_1} = 1$, $\vec{c_1} \cdot \vec{t_2} = 0$, and $\vec{c_1} \cdot \vec{t_3} = 0$. Therefore we have, $t_{11} = 1$, $t_{12} = 0$ and $t_{13} = 0$.

Step 1.1: Set $\vec{a_1} \cdot \vec{t_1} = 0$, $\vec{a_1} \cdot \vec{t_2} = 0$, and $\vec{a_1} \cdot \vec{t_3} = 1$. We have $\vec{a_2} \cdot \vec{t_3} = 0$. Therefore, $t_{31} = 0$, $t_{32} = 0$, and $t_{33} = 1$; and $\left(\vec{a_2} \cdot \vec{t_3} = 0\right) \Longrightarrow (t_{13} - t_{33} = 0)$, which is not true.

Step 1.2: Set $\vec{a_2} \cdot \vec{t_1} = 0$, $\vec{a_2} \cdot \vec{t_2} = 0$, $\vec{a_2} \cdot \vec{t_3} = 1$ and $\vec{a_1} \cdot \vec{t_3} = 0$. Therefore, $(t_{11} - t_{31} = 0) \Longrightarrow t_{13} = 1$; $(t_{12} - t_{32} = 0) \Longrightarrow t_{32} = 0$; and $(t_{13} - t_{33} = 1) \Longrightarrow t_{33} = -1$, which is not true since $t_{33} = 1$. Since the reference matrix for array A and B are the same, there can be no block transfers for B as well.

Step 2.0: (C column distributed) Set $\vec{c_2} \cdot \vec{t_1} = 1$, $\vec{c_2} \cdot \vec{t_2} = 0$, and $\vec{c_2} \cdot \vec{t_3} = 0$. Therefore, $(-t_{11} + t_{21} = 1) \Longrightarrow (t_{11} = t_{21} - 1)$, $(-t_{12} + t_{22} = 0) \Longrightarrow (t_{12} = t_{22})$, and $(-t_{13} + t_{23} = 0) \Longrightarrow (t_{13} = t_{23})$.

Step 2.1a: (First reference of A): Set $\vec{a_1} \cdot \vec{t_1} = 0$, $\vec{a_1} \cdot \vec{t_2} = 0$, and $\vec{a_1} \cdot \vec{t_3} = 1$. Therefore, $t_{31} = 0$, $t_{32} = 0$ and $t_{33} = 1$, and $\left(\vec{a_2} \cdot \vec{t_3} = 0\right) \Longrightarrow (t_{13} - t_{33} = 0) \Longrightarrow t_{13} = 1$ and $t_{23} = 1$.

Step 2.1b: (Second reference of A): Set $\vec{a_1} \cdot \vec{t_1} = 0$, $\vec{a_1} \cdot \vec{t_2} = 0$, and $\vec{a_1} \cdot \vec{t_3} = 1$. Therefore, $t_{31} = 0$, $t_{32} = 0$ and $t_{33} = 1$, and $(\vec{a_2} \cdot \vec{t_3} = 0) \Longrightarrow (t_{23} - t_{33} = 0)$ and $t_{23} = 1$.

Hence we have the inverse of the transformation matrix as $T^{-1} = \begin{pmatrix} t_{11} & t_{12} & 1 \\ t_{21} & t_{22} & 1 \\ 0 & 0 & 1 \end{pmatrix}$. Since the reference matrices for B is the same as that of A we distribute B and A in the same manner. We choose the unknown values such that T is a legal unimodular transformation. A possible T^{-1} is $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ and $T = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}$. The transformed reference matrices are as follows: $C'_{\mathcal{R}} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$, $A'^{,1}_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$, and $A'^{,2}_{\mathcal{R}} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$. Using the above algorithm we distribute the arrays A, B, and C by columns and the transformed code with block transfers is as follows:

for $u = 1$ to N do
  for $v = u$ to $min(u + 2b - 2, N)$ do
    read $A(*, v + b)$; read $A(*, u + v + b)$; read $B(*, u + v + b)$; read $B(*, v + b)$;
    for $w = max(u - b + 1, v - b + 1, 1)$ to $min(u + b - 1, v + b - 1, N)$ do
      $C[v + w + 1, u] += A[w, v + b] * B[w, u + v + b] + A[w, u + v + b] * B[w, v + b]$

5 Summary

This paper illustrated an algorithm which derives the terms in the transformation matrix which gives the best locality and minimum communication on distributed memory machines. We used the concept of *data reference matrices* for individual array references. Using this as the starting point, we systematically derived the best set of transformation matrices which give both good locality while enabling parallelism. Unlike [7], where a *padding matrix* is used along with an arbitrary set of rows in the *basis matrix,* we generate a transformation matrix systematically. The algorithm also gives an optimal distribution of arrays on to the processors such that block transfers are enabled to reduce inter-processor communication. Here, distribution of data only along one dimension is considered. However complex distributions with more than one distributed dimension can be derived using a simple extension of the above algorithm. Work is in progress in deriving more complex distribution of data and iterations along with tiling.

References

[1] J. Anderson and M. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *ACM SIGPLAN'93 PLDI,* June 1993, pp. 112–125.

[2] S. Chatterjee, J. Gilbert, R. Schreiber and S. Teng. Optimal evaluation of array expressions on massively parallel machines. RIACS Technical Report TR 92.17.

[3] M. Gupta. *Automatic data partitioning on distributed memory multicomputers.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Sept. 1992.

[4] K. Kennedy and U. Kremer. Automatic data alignment and distribution for loosely synchronous problems in an interactive environment. Tech. Rep. CRPC TR91-205, Rice University, April 1991.

[5] K. Knobe, J. Lukas and G. Steele Jr. Data optimization: Allocation of arrays to reduce communication on SIMD machines. *Journal of Parallel and Distributed Computing,* 8(2), Feb. 1990, 102–118.

[6] J. Li and M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of Parallel and Distributed Computing,*13(2) Oct. 1991, 213–221.

[7] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. *Proc. 5th Workshop on Languages and Compilers for Parallel Computing,* August 1992.

[8] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems,* 2(4):472–482, Oct. 1991.

[9] J. Ramanujam and A. Narayan. Automatic array distribution and loop transformations. Technical Report TR-94-07, Louisiana State University, January 1994.

[10] S. Wholey. *Automatic data mapping for distributed-memory parallel computers.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1991.