Data Alignment: Transformations to Reduce Communication on Distributed Memory Architectures

Michael O'Boyle and G.A. Hedayat
Department of Computer Science
Centre for Novel Computing
University of Manchester
Oxford Road Manchester M13 9PL, UK

Abstract

The relative storage, or alignment, of array data in distributed memory critically determines the amount of communication overhead. This paper expresses data alignment in a linear algebraic framework. Aligned data can be viewed as forming a hyperplane in the iteration space. This allows the quantification of data alignment and the determination of the existence of transformations to reduce non-local access. This has led to a new alignment transformation which is applicable to a wider class of problems than existing techniques. The global impact of such transformations are discussed as is the effect of alignment on partitioning.

1 Introduction

Compilation should minimise parallel time by utilising machine parallelism and reducing overhead. The first stage of compilation is therefore to identify and match program parallelism to machine parallelism. We define machine parallelism simply as the number of processors p. It is necessary to identify and divide a nested loop array computation into p sub-computations.

In compiling for distributed memory architectures (dmas), it is necessary to map the data and computation to processors so as to allow parallel evaluation. However in doing so it is possible that this will result in large overheads due to load imbalance and communication.

This paper is concerned with the reducing of overhead due to communication or non-local access. Reducing this overhead by making as many accesses local as is possible is often referred to as exploiting locality. There is in general a tension between parallelism and locality. Mapping all data and computation to one processor implies no communication overhead but also no parallelism. This paper attempts to minimise non-local access once machine parallelism has been utilised.

It is the mapping of data to processors that determines the amount of non-local access. As there are many ways to map any one array on to an array of processors, this is in general a complex problem.

To simplify this process it is convenient to break it into two sub-problems. All arrays are first mapped to a common array which is then partitioned across the processors. This paper covers the first stage known as alignment and is concerned with the relative orientation of arrays appearing in a computation.

After the mapping of data to processors, alignment determines which portion of one array will be in the same processor as another part of a second array. Obviously if this is done in such a manner that both portions are involved in the same computation, then this will reduce the number of non-local accesses that would have been incurred if they were in separate processors. Alignment is concerned with the relative allocation of arrays so as to maximise local accesses.

This approach to the mapping of data to processors has been used in several systems. Their alignment strategies can be broadly divided into those which rely on user pragmas and those based upon program analysis. This paper belongs to the second category and performs analysis and transformations based upon a matrix representation of array computation.

Good examples of pragma based alignment schemes include [2] and [10], where once a data partition for one array has been defined, others may be aligned to it. Automatic schemes include [3], where data allocation based upon the analysis of array occurrences has the effect of aligning arrays that reference each other. In [4], this process is more explicit; selected transformations such as permutation and shifts are used to improve alignment before data partitioning takes place.

The two major contributions of this paper are:

DO $j_1 = l_1, v_1$
  DO $j_2 = L_2(j_1) + l_2, U_2(j_1) + v_2$
    $\vdots$
    DO $j_m = L_m(j_1, \ldots, j_{m-1}) + l_m,$
      $U_m(j_1, \ldots, j_{m-1}) + v_m$
      $B_1[F_1(J)] = B_2[F_2(J)]$
        $\ldots B_\theta[F_\theta(J)]$

Figure 1: Parallel Nested Affine Loops

- the expression of alignment in a linear algebraic framework
- the derivation of a greater class of alignment transformations

Recent research has been aimed at finding a uniform theory of program transformations to reveal program parallelism [1], [9] and [11]. This framework has been used in [12] and [6] to detect opportunities for data re-use.

This paper presents alignment in a framework allowing a unifying approach to compiling for distributed memory architectures [7]. It is possible to determine whether an alignment transformation exists and the maximum amount of alignment possible. By using an alignment algorithm based upon a row echelon transformation, arrays may be transformed to give greater alignment.

2 Representation

In this paper we restrict our attention to parallel nested loops. The loop bounds and array references are restricted to the affine form described in figure 1. This is purely for the ease of presentation, in [7], general programs with arbitrary loop nests and conditional evaluation are considered.

The number of loops, occurring in a loop nest, is defined as m. L_k and U_k are integer linear combinations of the iterators $[j_1, j_2, \ldots, j_{k-1}]^T, 1 \leq k \leq m$ and l_k and v_k are lower and upper bound constants. The iteration vector J is defined over the range

$$LJ \geq l \tag{1}$$

$$UJ \leq v \tag{2}$$

where L and U are integer unit lower triangular matrices representing the loop bounds and l, v are integer vectors [8]. The system of inequalities can be re-expressed as

$$\begin{bmatrix} -L \\ U \end{bmatrix} J \leq \begin{bmatrix} -l \\ v \end{bmatrix} \tag{3}$$

This is often referred to as the *iteration space*, and will be denoted by V. For example the program in figure 2 has the following iteration space:

$$\left[\begin{array}{cc} -1 & 0 \\ 0 & -1 \\ \hline 1 & 0 \\ 0 & 1 \end{array}\right] \begin{bmatrix} i \\ j \end{bmatrix} \leq \left[\begin{array}{c} -1 \\ -1 \\ \hline n \\ n \end{array}\right] \tag{4}$$

The computation of array B_1 using array $B_2, \ldots, B_\theta$ makes affine references to the enclosing iterators.

The subscripts of an array reference to an array say B_i can be represented by $F_i(J)$ where

$$F_i^T(J) = [f_1^i(J), \ldots, f_{N_i}^i(J)] \tag{5}$$

N_i is the dimension of array B_i and $f_k^i(J), 1 \leq k \leq N_i$ is a function of the enclosing iterators. This paper restricts its attention to affine array references and hence F_i can be represented by a matrix/vector pair. In the example in figure 2

$$F_1(J) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6}$$

and

$$F_2(J) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7}$$

In general, array B_i references B_j, when

$$\hat{F}_i(J) = \hat{F}_j(J), 1 \leq, i, j \leq \theta \tag{8}$$

where

$$\hat{F_i}^T = [F_i^T, 0_{(M-N_i)\times m}] \tag{9}$$

$$\hat{F_j}^T = [F_j^T, 0_{(M-N_j)\times m}] \tag{10}$$

and $M = max(N_i, N_j, m)$. In other words $\hat{F_i}$ and $\hat{F_j}$ have extra zero rows added to them so as to make their sizes compatible.

If we define $S = \{J \in Z^m : \hat{F}_i(J) = \hat{F}_j(J)\}$, then S is a hyperplane in Z^m and $S \cap V$ denotes the points in iteration space where references to array B_j by array B_i are aligned. For example, consider the program in figure 2:

$S = \{i, j \in Z^2 : i = j, j = i\}$ and $S \cap V = \{i \in [1..n], j \in [1..n] : i = j, j = i\}$, and $|S \cap V| = n$. This means that in program 2 of the n^2 access to b by array a, n of them are aligned and will be local accesses whatever data partitioning is employed. However after data partitioning, the number of non-local accesses will be $O(n^2)$ which implies a large overhead.

```
DO i = 1, n
   DO j = 1, n
         a(i, j) = b(j, i)
```

Figure 2: Simple Transpose

Alignment of B_j with B_i, can then be formulated as an optimization problem as follows: Find transformation T such that

$$|S \cap V| \tag{11}$$

is maximized. where $S = \{J \in Z^m : \hat{F}_i(J) = T\hat{F}_j(J)\}$,

In program 2 if T is chosen as:

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, TF_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \tag{12}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \tag{13}$$

$S = Z^m$ and $|S \cap V| = |V| = n^2$ which implies that for any data partition there will be n^2 local access and hence no non-local access regardless of any subsequent data partition.

Even for pairwise alignment the above optimization problem, in general involves enumerating the integer set $S \cap V$ for each candidate T, a difficult task. This leads us to consider a suboptimal procedure, which is amenable to polynomial time determination of the transformation T

3 Transformation

We first define the class of alignment transformation $T = (\mathcal{A}, a)$ for a pair of arrays $B_i[F_i(J)]$ and $B_j[F_j(J)]$. This can be extended to multiple arrays.

Let $F_i(J) = \mathcal{C}J + c$, and $F_j(J) = \mathcal{U}J + u$, where $\mathcal{C}$, $\mathcal{U}$ are $N_i \times m$ and $N_j \times m$ occurrence matrices, and c, u are occurrence vectors of arrays B_i and B_j respectively. Both occurrence/vector pairs are padded to make their sizes compatible.

$$\hat{\mathcal{U}}^T = [\mathcal{U}^T, 0_{(M-N_i)\times m}] \tag{14}$$

and

$$\hat{\mathcal{C}}^T = [\mathcal{C}^T, 0_{(M-N_j)\times m}] \tag{15}$$

The transforms considered are restricted to affine bijective alignment transformations i.e.

$$T : \hat{\mathcal{U}}J + \hat{u} \mapsto \mathcal{A}\hat{\mathcal{U}}J + a + \hat{u} \tag{16}$$

where $\mathcal{A}$ is an $M \times M$ invertible matrix and a is an M integer shift vector.

4 Measure of alignment

In this section, a suboptimal criterion to determine the alignment transformation T for a pair of arrays $B_i[\mathcal{C}J + c]$, and $B_j[\mathcal{U}J + u]$ is proposed

$H_{\mathcal{U},\mathcal{C}}$ is defined to be the number of perfectly aligned non-zero rows of the occurrence matrices of the arrays B_i, B_j For example in the assignment

$$B_i[i, i+j, i+k] = B_j[j, i+j, j] \tag{17}$$

where the occurrence matrices can be represented as:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \tag{18}$$

the value of H is one as only the second indices are identical. It can be shown that

$$H_{\mathcal{U},\mathcal{C}} = dim\ S = dim\{J \in Z^m : \hat{\mathcal{U}}J + \hat{u} = \hat{\mathcal{C}}J + \hat{c}\} \tag{19}$$

It is the purpose of any applied transform $\mathcal{A}$ to maximise $H_{T\mathcal{U},\mathcal{C}}$. The conditions for existence of such a transformation are determined, and the maximum value of H for a given computation is derived.

5 Existence and uniqueness condition

The optimal shift vector a can always be chosen independently.

$$a = \hat{c} - \hat{u} \tag{20}$$

Therefore in order to maximise alignment in the sense of H, it is necessary to find as many solutions, y_i, to

$$y_i\hat{\mathcal{U}} = \hat{\mathcal{C}}_i \tag{21}$$

where y_i and $\hat{\mathcal{C}}_i$ are the ith row of the matrix $\mathcal{A}$ and $\hat{\mathcal{C}}$ respectively. In this sections we consider the issues of existence, uniqueness which are based on well known results from linear algebra.

Equation 21 can be written in the more familiar form:

$$\hat{\mathcal{U}}^T y_i^T = \hat{\mathcal{C}_i}^T \tag{22}$$

where $y^T, \hat{\mathcal{C}_i}^T$ are column vectors.

The following results can be shown

- It is possible to align a row of $\mathcal{U}$ with $\mathcal{C}$ iff $\mathcal{C}_i^T$ is perpendicular to the null space of $\mathcal{U}$ i.e. $\mathcal{C}_i^T \perp \mathcal{N}(\mathcal{U})$.
- The solution to $\hat{\mathcal{U}}^T y_i^T = \hat{\mathcal{C}_i}^T$ is unique iff $dim\mathcal{N}(\hat{\mathcal{U}}^T) = 0$
- The maximum value of $H_{T\mathcal{U},\mathcal{C}}$ is $\min(r, d)$ where $r = rank(\mathcal{U})$ and $d = rank(\mathcal{C})$

6 Row echelon form

$\mathcal{A}$ is chosen to maximize the number of perfectly aligned indices of $\mathcal{A}\hat{\mathcal{U}}$ and $\hat{\mathcal{C}}$. In order to determine the form of $\mathcal{A}$, such that $H_{\mathcal{AU},\mathcal{C}}$ is maximised, it is necessary, to find a solution for as many rows of $\mathcal{C}$ to the following equation:

$$\hat{\mathcal{U}}^T y_i^T = \hat{\mathcal{C}_i}^T \tag{23}$$

To solve for y_i^T, concatenate $\hat{\mathcal{U}}^T$ and $\hat{\mathcal{C}_i}^T$ and reduce $\hat{\mathcal{U}}^T$ by elementary row operations to row echelon form, whilst simultaneously performing the same operations on $\hat{\mathcal{C}}_i^T$. Equation 23 is re-written as:

$$\left[\begin{array}{c|c} \hat{\mathcal{U}}^T & \hat{\mathcal{C}_i}^T \\ m \times M & m \times 1 \\ \hline O & O \\ M-m \times M & M-m\times 1 \end{array}\right] \tag{24}$$

Reducing by elementary row operations, gives the unique row-echelon form:

$$\left[\begin{array}{cc|c} Re & * & \hat{\mathcal{C}}_i^1 \\ r \times r & r \times M-r & r \times 1 \\ \hline O & O & \hat{\mathcal{C}}_i^2 \\ m-r\times r & m-r \times M-r & m-r \times 1 \\ \hline O & O & O \\ M-m \times r & M-m \times M-r & M-m\times 1 \end{array}\right] \tag{25}$$

Here Re and $*$ are non-zero sub-matrices.

6.1 Algorithm

There is a solution iff $\mathcal{C}_i^T \perp \mathcal{N}(\mathcal{U})$. This is equivalent to $\hat{\mathcal{C}}_i^2 = 0$ which provides a simple existence test. If a solution exists then it forms a row of $\mathcal{A}$ and as there are at most $\min(r,d)$ solutions, the remaining rows of $\mathcal{A}$ have to be filled.

This algorithm reduces an $m \times M$ matrix $\hat{\mathcal{U}}^T$ to row echelon form while simultaneously carrying the same elementary operations on $m \times M$ array $\hat{\mathcal{C}}^T$ and the identity matrix I_M:

1. Reduce
$$\left[\begin{array}{c|c|c} \hat{\mathcal{U}}^T & \hat{\mathcal{C}}^T & \\ \hline O & O & \end{array} \; I_M \right] \tag{26}$$
to row echelon form
2. Reorder by elementary row operations such that leading non-zeros lie on the diagonal
3. If $\hat{\mathcal{C}}_i^2 = 0$ and $\hat{\mathcal{C}}_i$ is independent then place in row i of $\mathcal{A}$ otherwise select an independent row of I_M

```
DO i = 1,n+3
  DO j = 1,2n
    b[i,j] = i+j
DO i = 1,n
  DO j = 1,n
    c[i,j] = i-j
      .
      .
      .
DO i = 1,n
  DO j = 1,n
    DO k = 1,n
      a[i,j] += b[j+3,i+j] * c[k,n-i+1]
```

Figure 3: Example Program

Although $\mathcal{A}$ may in general be a matrix with rational elements, $\mathcal{A}\hat{\mathcal{U}}$ is always integer. Thus for pairwise alignment, the row echelon form gives the required alignment transformation. Consider the calculation of the array a in the program segment in figure 3:

The arrays, a, b, c, have the following occurrence matrix/vectors :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}_a \begin{bmatrix} i \\ j \\ k \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}_b \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \tag{27}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_c \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ n+1 \\ 0 \end{bmatrix} \tag{28}$$

As long as the occurrence matrices refer to different arrays, they may be compared pair-wise with the common array. Thus for aligning b with a after reducing to row-echelon form we have:

$$\left[\begin{array}{ccc|ccc|ccc} 1 & 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \tag{29}$$

All leading non-zero elements are on the diagonal and the first columns of the $\mathcal{C}$ region provide independent solutions, while the remaining row of $\mathcal{A}$ is selected from the I region. This gives

$$\mathcal{A} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} a = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} \tag{30}$$

Which gives the following new occurrence matrix:

```
        .
        .
        .
DO i = 1,n
  DO j = 1,n
    DO k = 1,n
      a[i,j] += b[i,j] * c[i,k]
```

Figure 4: Aligned Program Segment

$$\mathcal{A} \times \hat{\mathcal{U}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} a + \hat{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad (31)$$

Repeating this procedure for array c we have:

$$\left[\begin{array}{ccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \qquad (32)$$

Giving

$$\mathcal{A} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} a = \begin{bmatrix} 0 \\ -n-1 \\ 0 \end{bmatrix} \qquad (33)$$

Which gives the new occurrence matrix :

$$\mathcal{A} \times \hat{\mathcal{U}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} a + \hat{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad (34)$$

Therefore the calculation of a may be written as shown in figure 4:

Before the alignment transformations the value of $H_{b,a}$ and $H_{c,a}$ was 0. After the alignment transformations, $H_{b,a} = 2$ and $H_{c,a} = 1$. If the array a is now partitioned along the first index, then the subsequent implementation will have no non-local access and hence no overhead due to communication. This program transformation provides the relative allocation of data so as to achieve maximum data alignment. Such an allocation strategy would not be possible using existing automatic techniques [4],[3].

In this example the construction of $\mathcal{A}$ was relatively straightforward, difficulties occurr if the array to align to is rank deficient as two solutions may not be independent. In this case, one solution is chosen and an independent column from the identity region selected.

```
DO i = 1,n+3
  DO j = 1,2n
    b[j-i+3,i-3] = i+j
DO i = 1,n
  DO j = 1,n
    c[-j+n+1,i] = i-j
        .
        .
```

Figure 5: Propagated Alignment

7 Propagation of alignment

Once alignment has been determined for an array, all other occurrences of that array variable must be aligned accordingly throughout the program to maintain meaning. All other occurrences of an array must be transformed by $(\mathcal{A}, \mathcal{A} \times a)$. Before alignment, the occurrences matrix/vector of b, when created, was

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}_b \qquad (35)$$

After alignment:

$$\mathcal{A}_b \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}_b = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad (36)$$

and

$$\mathcal{A}_b \times a_b + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_b = \begin{bmatrix} 3 \\ -3 \\ 0 \end{bmatrix} \qquad (37)$$

A similar procedure is applied to c to give the program in figure 5.

The new occurrences must all be integer matrices so this in general restricts $\mathcal{A}$ to class of integer matrices. Currently work is in progress on generation of integer alignment transformations using Hermite and Smith normal form of array occurrences.

It is easy to see that if an array is accessed by two or more arrays, they will, in general, have different references. Each usage of an array may suggest a conflicting alignment. Therefore choosing the alignment of arrays is important as conflicting requirements have to be satisfied which has a global effect throughout the program. Conflict resolution heuristics are beyond the scope of this paper but several schemes have been proposed in [3],[4],[7].

8 Data partition

Once the relative alignment between arrays is determined, the arrays must be mapped to processor space. At this stage only orthogonal partitions of the arrays are considered. Thus if an array has N dimensions, it may be partitioned in $2^N - 1$ different ways. If possible, the partitioning of data should be along those dimensions of the array that may be evaluated in parallel. In general, however, the decision on how to partition data will be based on several criteria [7].

If two arrays are aligned with respect to a particular index, then no matter how those individual array elements are partitioned, any reference between the two arrays with respect to this index will always be local. Non-aligned indices do not have this property and thus partitioning with respect to these indices should be avoided. If an array has N dimensions and say k of them are aligned, then there are k dimensions the array may be partitioned along, and $N-k$ that should be *serialised*. As the size of any one dimension of an array is assumed to be greater than the number of processors i.e. $n >> p$, it is reasonable to consider only one dimension to partition along at this stage. Partitioning an array is the act of projecting an $M \times m$ occurrence matrix so that there is only one non-zero row in the new occurrence matrix, which corresponds to that dimension of the array being distributed over the p processors.

$\mathcal{P}$ and $\mathcal{S}$ are the partitioning and serialising matrices where

$$\mathcal{P} + \mathcal{S} = I \tag{38}$$

and $\mathcal{P} \times \mathcal{C}$ defines the dimension of $\mathcal{C}$ to be distributed across the processors. Unlike the alignment matrix $\mathcal{A}$ these transformation matrices are formed to reduce the rank of the occurrence matrices and thus are singular.

$\mathcal{P}$ should be chosen so as to maximise

$$\sum_{i=2}^{\theta} H_{\mathcal{PC},\mathcal{PU}_i} \tag{39}$$

for all the arrays. In the previous program, figure 4, $\mathcal{P}$ is chosen to be of the following form:

$$\mathcal{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{40}$$

which gives the following reduced occurrence matrices:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_a, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_b, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_b \tag{41}$$

The sum of $H_{\mathcal{PC},\mathcal{PU}}$ is 2 which is the maximum value possible. If there are $\theta - 1$ arrays to be aligned, the maximum value of $\sum_{i=2}^{\theta} H_{\mathcal{PC},\mathcal{PU}_i}$ is $\theta - 1$. This implies that there will be no non-local access.

This observation gives a simple method of deciding which index to partition along and hence a method for constructing $\mathcal{P}$. Let the kth row of $\mathcal{P}$ be the kth row of Identity e_k^T and the remaining rows set to zero. k is chosen to maximise:

$$\sum_{i=2}^{\theta} H_{e_k^T \mathcal{C}, e_k^T \mathcal{U}_i} \tag{42}$$

The determination of $\mathcal{P}$ is an $O(M \times \theta)$ operation.

In those cases where the arrays to be partitioned have a different number of dimensions and where arrays are padded further analysis is required, for more details of this analysis see [7].

Choosing $\mathcal{P}$ in this manner will maximise the amount of useful data local to a processor. However, other considerations such as load balancing and maximising data re-use may require a different data partitioning strategy.

9 Conclusion

By expressing array references in a linear algebraic framework, it has been shown that aligned data forms a hyperplane in the iteration space. This allows the quantification of data alignment and the determination of the existence of transformations to reduce non-local access. The row echelon transformation used is applicable to a wider class of problems than existing techniques. Once a particular alignment has been chosen, its effect must be propagated throughout the rest of the program. The work presented in this paper forms part of the general mapping of array computation to distributed memory architectures. In [7] matrix transformations to perform load balancing, data partitioning and data re-use are also expressed within a unified framework. Current work involves the investigation of the trade off between parallelism, non-local access and load imbalance and the prototype implementation of such a scheme.

Acknowledgement

This work is supported by the Science and Education Research Council, UK, grant number GR/FO4292.

References

[1] Banerjee U.,**Unimodular Transformations of Double Loops**, Proc. of 3rd Workshop on Programming Languages and Compilers for Parallel Computing, Irvine CA, August 1990.

[2] Fox G., Hiranandani S., Kennedy K., Koelbel C., Kremer U., Tseng C-W. and Wu M-Y., **"FORTRAN D Language Specification"**, Rice COMP TR90-141, Department of Computer Science, Rice University, February 1991.

[3] Knobe K., Lukas J.D., and Steele G.L., **"Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines"**, Journal of Parallel and Distributed Computing 8, pp102-118 1990.

[4] Li J. and Chen M., **"Index Domain Alignment: Minimising Cost of Cross-Referencing between Distributed Arrays"**, IEEE Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation, October 1990.

[5] Lu L-C., **"A Unified Framework for Systematic Loop Transformations"**, 3rd ACM Symposium on the Principles and Practice of Parallel Programming, April 1991.

[6] O'Boyle M.F.P., Hedayat G.A., **"A New Program Transformation to Minimise Communication in Distributed Memory Architectures"**, accepted for PARLE'92, Paris, June, 1992.

[7] O'Boyle M.F.P., **"Program and Data Transformations for Efficient Execution on Distributed Memory Architectures"**, PhD thesis, University of Manchester, Dept. of Computer Science, January 1992.

[8] O'Boyle M.F.P., Hedayat G.A., **"Load Balancing of Parallel Affine Loops by Unimodular Transformations"**, The European Workshop on Parallel Computing, Barcelona, March, 1992.

[9] Pugh W., **"Uniform Techniques for Loop Optimisation"**, International Conference on Supercomputing, June 1991.

[10] Rogers A., **"Compiling for Locality of Reference"**, Technical Report TR 91-1195, Department of Computer Science, Cornell University, March 1991.

[11] Wolf M.E. and Lam M.,**"An Algorithmic Approach to Compound Loop Transformations"**, in *Advances in Languages and Compilers for Parallel Processing"*, eds Nicolau A., Gelernter D., Gross T. and Padua D., Pitman London, 1991.

[12] Wolf M.E. and Lam M.,**"A Data Locality Optimizing Algorithm"**, ACM SIGPLAN Conference on Programming Language Design and Implementation, June 1991.