Compiling affine nested loops: how to optimize the residual communications after the alignment phase?

Michle Dion, Cyril Randriamaro and Yves Robert *

Laboratoire LIP, URA CNRS 1398

Ecole normale suprieure de Lyon, F-69364 LYON Cedex 07

e-mail: [Michele.Dion,Cyril.Randriamaro,Yves.Robert]@ens-lyon.fr

Abstract

Minimizing communication overhead when mapping affine loop nests onto distributed memory parallel computers (DMPCs) is a key problem with regard to performance, and many authors have dealt with it. All communications are not equivalent. Local communications (translations), simple communications (horizontal or vertical ones), or structured communications (broadcasts, gathers, scatters, or reductions) are performed much faster than general affine communications onto DMPCs.

In this paper, we recall the mapping heuristic given by Dion and Robert which consists in minimizing the number of nonlocal communications and we focus on the next step: as it is generally impossible to obtain a communication local mapping, we show how to optimize residual general communications using structured communications or decompositions into small sequences of simple communications.

1 Introduction

This paper deals with the problem of mapping affine loop nests onto Distributed Memory Parallel Computers (DMPCs). Because communication is very expensive in DMPCs, how to distribute data arrays and computations to processors is a key factor to performance.

*Supported by the CNRS—ENS Lyon–INRIA project *ReMaP* and by the *LHPC EuroTOPS* project.

The computations described in this paper are general non-perfect loop nests (or multiple loop nests) with uniform or affine dependences. Mapping such loop nests onto virtual processors with the aim of "minimizing" in some sense the communication volume or number is known as the alignment problem, which has drawn a lot of attention [16, 10, 14, 1, 4, 20, 13, 3, 7].

More precisely, given a loop nest, the goal is to assign a virtual processor, i.e. a location on a virtual processor grid, to each array element and to each computation. Arrays and computations are thus aligned and projected onto the virtual processor grid. This grid is then folded onto a physical grid, most often of much smaller size in each dimension. Languages like *HPF* provide `BLOCK` or `CYCLIC` distributions for such a folding.

There is a natural objective function for the mapping problem, which has been extensively used in the literature: the aim is to minimize the number of nonlocal communications that will remain after a suitable alignment has been found. Some authors even look for a *communication free* mapping for which there remains *NO* nonlocal communication. In fact, it is always possible to achieve a communication free mapping ... provided we are prepared to use a virtual grid of dimension 0, i.e. a single processor ! Obviously, the larger the dimension of the target virtual grid, the larger the number of residual nonlocal communications.

Experimental evidence shows that communication free mappings are very unlikely to be achieved. Think of elementary kernels as simple as a matrix-matrix product or a Gaussian elimination procedure, there is no way to map such kernels onto 2D- or even 1D-grids without residual general communications. A natural question arises: is there a way to "optimize" in some sense the communications that remain ?

Platonoff [19] gives a strong motivation to answer the question. Experimenting on a $CM-5$ with 32 processors, he compared various communication times. He observed the ratios reported in Table 1. Table 1 clearly shows that the $CM-5$ has facilities for some usual structured communications such as broadcasts or reductions. Also, translations are much more efficient than general (affine) communications.

Experimenting with the Intel Paragon gives interesting results too: communication conflicts are generated by serial messages on a single link at the same time. Thus a general communication cannot be efficiently executed by simply letting all processors send their messages simultaneously. Decomposing a general communication into a small sequence of communications parallel to the grid axes (horizontal and vertical) proves much faster, as

Reduction	Broadcast	Translation	General communication
1	1.5	2.5	90

Table 1: Comparing execution times for different data movements on a CM-5

illustrated in Section 4.1.

Being prepared to have some residual communications, we adopt the following strategy:

1. Zero out as many nonlocal communications as possible. To this purpose, we use an heuristic modified from Dion and Robert [7]. This heuristic is based upon the *access graph* and will be explained in Section 2. We weight the edges of the access graph to give different priorities to the communications according to a given criteria.

2. For residual communications, we explore the following two possibilities (both can be implemented simultaneously):

   (a) try to find a mapping such that (at least) one of the residual communications is a structured communication.

   (b) try to find a mapping such that (at least) one of the residual communications can be decomposed into more simple and efficient data movements (such as horizontal or vertical communications).

The paper is organized as follows: in Section 2, we introduce a motivating example that we will work out throughout the paper. We informally explain our modified heuristic to zero out as many communications as possible. We explain how to take advantage of structured communications in Section 3. We explain how to decompose general affine communications into simpler ones in Section 4. Altogether, our complete heuristic for the mapping problem is summarized in Section 5. In Section 6, we give some results of the heuristic on classical linear algebra examples. Section 7 is devoted to a survey of related work in the literature, together with some discussion and comparisons. Finally, concluding remarks are stated in Section 8.

2 Motivating example

In this section, we informally explain our new approach on an example. First we apply an heuristic modified from [7] to minimize the number of nonlocal communications. Then, we try to optimize the residual general communications using structured communications or decompositions.

2.1 Example

Consider the following non-perfect affine loop nest, where a is a $2D$-array, and b and c are $3D$-arrays:

Example 1

for $i = 0$ to N do

  for $j = 0$ to M do

    { Statement S_1 } $b(F_1 \cdot (i,j)^t + c_1) = g_1(a(F_2 \cdot (i,j)^t + c_2), a(F_3 \cdot (i,j)^t + c_3), c(F_4 \cdot (i,j)^t + c_4))$

    for $k = 0$ to $N + M$

      { Statement S_2 } $b(F_5 \cdot (i,j,k)^t + c_5) = g_2(a(F_6 \cdot (i,j,k)^t + c_6))$

      { Statement S_3 } $c(F_7 \cdot (i,j,k)^t + c_7) = g_3(a(F_8 \cdot (i,j,k)^t + c_8))$

endfor

where $F_1 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & -1 \end{pmatrix}$ and $c_1 = \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix}$, $F_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $c_2 = \begin{pmatrix} -1 \\ 6 \end{pmatrix}$, $F_3 = \begin{pmatrix} 1 & 1 \\ -4 & -5 \end{pmatrix}$ and $c_3 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $F_4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{pmatrix}$ and $c_4 = \begin{pmatrix} -1 \\ 3 \\ 0 \end{pmatrix}$, $F_5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = Id_3$ and $c_5 = \begin{pmatrix} 2 \\ 5 \\ -3 \end{pmatrix}$, $F_6 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$ and $c_6 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$, $F_7 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$ and $c_7 = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$, $F_8 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ and $c_8 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

Here, g_1, g_2 and g_3 are arbitrary functions. The loop nest is an *affine loop nest* because all array references are affine functions of the loop indices. There are no data dependences in the nest (check this with Tiny [22] for instance), all loops are `DOALL` loops, hence all computations can be executed at the same time-step.

Mapping the loop nest onto a m-dimensional virtual processor space consists in determining an allocation matrix for each statement and for each array. *Affine allocation functions* are defined as in [1, 2, 7]:

1. For each statement S of depth d: $\text{alloc}_S(I) = M_S I + \alpha_S$, where I is the iteration vector ($I = (i,j)^t$ for statement S_1 and $I = (i,j,k)^t$ for statements S_2 and S_3), M_S is a $m \times d$ matrix and α_S is a m-vector.

2. For each array x of dimension q_x: $\text{alloc}_x(I) = M_x I + \alpha_x$, where M_x is a $m \times q_x$ matrix and α_x is a m-vector.

By this mapping, statement instance $S(I)$ is assigned to virtual processor $M_S I + \alpha_S$ and array element $x(I)$ is stored in the local memory of processor $M_x I + \alpha_x$. For instance, in statement S_1, the value $a(F_2(i,j)^t + c_2)$ has to be read in the memory of the processor

$$\text{alloc}_a(F_2(i,j)^t + c_2) = M_a(F_2(i,j)^t + c_2)I + \alpha_a$$

and sent to the processor in charge of the computation $S_1((i,j)^t)$, namely processor

$$\text{alloc}_{S_1}((i,j)^t) = M_{S_1}(i,j)^t + \alpha_S.$$

This results in a communication of length δ_{a,S_1} equal to the distance between $a(F_2(i,j)^t + c_2)$ and $S_1(i,j)$. We have

$$\delta_{a,S_1} = \text{alloc}_{S_1}((i,j)^t) - \text{alloc}_a(F_2(i,j)^t + c_2)$$
$$\delta_{a,S_1} = (M_{S_1} - M_a F_2)(i,j)^t - M_a c_2 + \alpha_{S_1} - \alpha_a.$$

The expression δ_{a,S_1} contains a *nonlocal* term $(M_{S_1} - M_a F_2)(i,j)^t$ which depends upon the iteration vector $(i,j)^t$, and a *local* term $\alpha_{S_1} - M_a c_2 - \alpha_a$. The *nonlocal* term corresponds to irregular patterns of communication, whose size can grow over the whole processor space. It is clearly the main factor affecting performance. On the other hand, the *local* term corresponds to regular fixed-size communications that can be performed efficiently onto DMPCs.

Zeroing out the nonlocal term is of course the main goal of the mapping optimization process, as recognized by [1, 2, 20]. A *communication local* mapping is a mapping where all nonlocal terms have been zeroed out. However, it is generally impossible to obtain a *communication local* mapping with the required dimension for the target virtual architecture. Then, another objective of the mapping process is to make efficient those nonlocal communications that cannot be zeroed out. We will try for example to derive structured communications such as broadcasts, scatters, gathers, reductions or to decompose general communications into communications parallel to the axes of the target processor space.

Back to our example, in statement S_1 and for array a, we see that zeroing out the nonlocal term amounts to choose allocation matrices M_{S_1} and M_a so that the equation $M_{S_1} - M_a F_2 = 0$ is satisfied. In the same way, the value $b(i+j-2, 1, 3-j)$ has to be written after the computation $S_1((i,j)^t)$. This results in a communication:

$$\delta_{S_1,b} = \text{alloc}_b(F_1(i,j)^t + c_1) - \text{alloc}_{S_1}((i,j)^t)$$
$$\delta_{S_1,b} = (M_b F_1 - M_{S_1})(i,j)^t + M_b c_1 + \alpha_b - \alpha_{S_1}$$

Again, to zero out the nonlocal term, we have to fulfill equation $M_b F_1 - M_{S_1} = 0$.

More generally, if an array x is read or written in a statement S with an access matrix F, to zero out the nonlocal part of the communication due to this access, we must satisfy the matrix equation $M_S = M_x F$. There are 8 such equations in our example.

2.2 Zeroing out nonlocal communications

The primary goal is to zero out as many nonlocal terms as possible. That is to say: given M_S (resp. M_x) of full rank[1], we want to find M_x (resp. M_S) of full rank such as $M_S = M_x F$. How to solve such an equation is explained below.

2.2.1 Solving $M_S = M_x F$

Consider the nonlocal term linking a statement S of depth d and an array x of dimension q_x: the equation is $M_S = M_x F$, where M_S is a $m \times d$ matrix, and M_x a $m \times q_x$ matrix. The matrix F is of dimension $q_x \times d$. If F is of deficient rank, we cannot find a matrix M_S of full rank such that $M_S = M_x F$. Hence, we consider only full rank access matrices.

[1]We search for full rank allocation matrices. Otherwise, the target m-dimensional processor space would not be fully utilized.

We target a m-dimensional processor space and we assume that $m \leq d$ and $m \leq q_x$ (we have chosen to deal only with the communications such that $m \leq d$ and $m \leq q_x$ as they represent the core of the computations and data elements to be distributed). As already said, we impose that the matrices M_S and M_x are of full rank m to fully utilize processor resources. There are several cases according to the shape of the matrix F:

$q_x = d$ (in such a case F is square).

This is clearly the simplest case when F is square and non-singular. Given M_x of rank m, then $M_x F$ is of rank m and we can let $M_S = M_x F$ without violating the constraint that M_S is of rank m. Conversely, given M_S of rank m, $M_S F^{-1}$ is of rank m, and we can let $M_x = M_S F^{-1}$.

$q_x < d$ (in such a case, F is flat).

Assume that F is of full rank. Then, given M_x of rank m, $M_x F$ is of rank m (see [6]), and we can safely let $M_S = M_x F$. However, given M_S, finding M_x of rank m such that $M_S = M_x F$ is not always possible. We know that F admits a right pseudo-inverse (see [6]). F^{-1} of size $d \times q_x$ and of rank q_x, such that $FF^{-1} = Id$. Hence, if there exists M_x such that $M_S = M_x F$, then $M_S F^{-1} = M_x F F^{-1} = M_x$. Unfortunately, $M_x = M_S F^{-1}$ is not always a solution of the equation $M_S = M_x F$, we have the compatibility condition[2] $M_S = M_S F^{-1} F$. Furthermore, $M_S F^{-1}$ can be of arbitrary rank less than m. To summarize, given M_x of rank m, it is always possible to determine M_S of rank m ($M_S = M_x F$) while the converse is not true.

$d < q_x$ (in such a case, F is narrow).

Assume that F is of full rank d. Then, F has a left pseudo-inverse F^{-1} of size $q_x \times d$, of rank d, and such that $F^{-1}F = Id$. The situation is exactly the converse of the previous one: given M_S of rank m, $M_x = M_S F^{-1}$ is a rank-m solution (see [6]) of the equation $M_S - M_x F = 0$; however, given M_x of rank m, $M_x F$ can be of arbitrary rank less than m. Hence, given M_S of rank m, it is always possible to determine M_x of rank m ($M_x = M_S F^{-1}$) while the converse is not true.

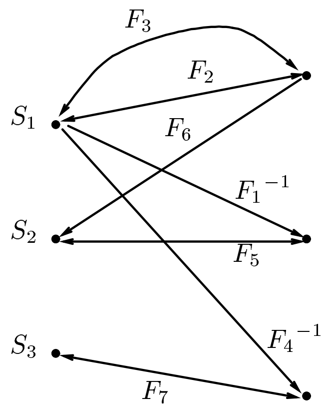

Figure 1: Access graph ($m = 1$ or 2)

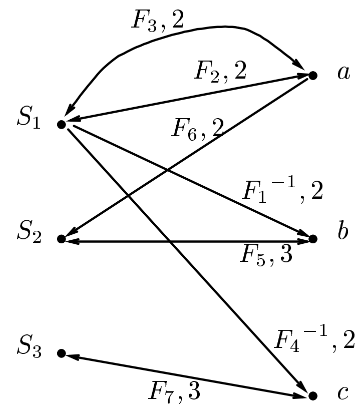

Figure 2: Access graph with integer weights

2.2.2 Access graph

We recall here the definition of a m-dimensional access graph $G = (V, E, m)$ graph as stated in [7]. The access graph takes into account how the equation $M_S = M_x F$ can be solved.

1. m is the dimension of the target virtual architecture,
2. each vertex $v \in V$ represents an array variable or a statement,
3. consider a loop nest in which an array variable x of dimension q_x is accessed (read or written) in a statement S of depth d, through an access matrix F of rank $\min(q_x, d)$ greater than the dimension m of the target architecture: then if $q_x \leq d$ we have an edge from x to S, to indicate that given M_x of rank m it is always possible to find M_S of rank m such that the communication is local, the matrix weight of the edge is F (in the following, we will also associate an integer weight to each edge); and if $d \leq q_x$ we have an edge from S to x, to indicate that given M_S of rank m it is always possible to find M_x of rank m such that the communication is local, the matrix weight of the edge is F^{-1}. In the special case where $q_x = d$, there is a single edge with two arrows between S and x to indicate that both orientations are possible.

Figure 1 represents the access graph for our example. The communication corresponding to the access matrix F_8 is not represented because F_8 is not a full rank matrix.

Remark For a narrow $q_x \times d$ matrix F, the left pseudo-inverse is not the only matrix G that satisfies the equation $GF = \mathrm{Id}$. Any matrix H such that $H = F^{-1} + M(\mathrm{Id} - FF^{-1})$

where M is an arbitrary matrix of correct dimension also satisfies $GF = \text{Id}$. Hence, in the access graph, we can choose any full rank matrix G such that $GF = \text{Id}$ as weight matrix (and not necessarily the "true" left pseudo-inverse F^{-1} as defined in [15]).

2.2.3 Mapping heuristic

Some nonlocal communications represented in the graph can be zeroed out, but not all. Consider a simple path in the access graph going from vertex v_1 to vertex v_2 with $v_1 \neq v_2$ (the path is not a cycle). Then given any allocation matrix M_{v_1} of rank m for vertex v_1, the existence of the path ensures that it is always possible to make local all communications represented by edges between the two vertices. In our example, given M_a of rank 2 in $G = (V, E, 2)$, and following the path $a \longrightarrow S_1 \longrightarrow b \longrightarrow S_2$, we are ensured to be able to compute M_{S_1}, M_b and M_{S_2}, all of rank 2, so that the communications corresponding to access matrices F_2, F_1 and F_5 are made local: we successively let $M_{S_1} = M_a F_2$, $M_b = M_{S_1} F_1^{-1}$ and $M_{S_2} = M_b F_5$.

There is another path from a to S_2 with the direct edge $a \longrightarrow S_2$ (access matrix F_5 for reading a in S_2). Is it possible to make local this communication in addition to the three above communications ? Using the edge $a \longrightarrow S_2$ we get the equation $M_{S_2} = M_a F_6$ while we had $M_{S_2} = M_b F_5 = M_{S_1} F_1^{-1} F_5 = M_a F_2 F_1^{-1} F_5$.

To simplify the equations, in the following we use the matrix $F_1' = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}$ (which satisfies the equation $GF_1 = \text{Id}$) instead of $F_1{}^{-1}$. We derive the condition $M_a F_2 F_1' F_5 = M_a F_6$. This condition is satisfied for all matrices M_a of rank m iff $F_2 F_1' F_5 = F_6$. In our example, let $F_{path} = F_2 F_1' F_5$. We compute $F_{path} = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix} \neq F_6$.

Note that as $F_{path} - F_6 = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ is of deficient rank, according to m, it will or not be possible to find a matrix M_a such that the condition $M_a F_2 F_1' F_5 = M_a F_6$ is satisfied.

In fact, this analysis can be extended in the general case: each time there are two disjoint paths p_1 and p_2 both going from a vertex v_1 to a vertex v_2 in the access graph, we can make all communications on both paths local provided that the equality $F_{p_1} = F_{p_2}$ holds (where F_p denotes the product of the access matrices along the edges of path p). If $F_{p_1} - F_{p_2}$ is of deficient rank, according to the size of the allocation matrix, it can or not be possible to find a matrix M such that $M(F_{p_1} - F_{p_2}) = 0$. See [7] for more details.

Besides, a similar analysis can be performed in the case of cycles. Denote by F_c the product of the weight matrices along the cycle: if $F_c = \text{Id}$ (where Id is the identity matrix), all the communications can be made local along the cycle, if $F_c - \text{Id}$ is of deficient rank, according to the size of the allocation matrix, it can or not be possible to have only local communications along the cycle.

As already said, the access graph depends upon the dimension m of the target architecture space. Given m, not all the communications are taken into account in the access graph $G = (V, E, m)$. The edges in G represent only the communications with access matrix of full rank greater than m. So, the heuristic does not try to make all the communications local but only the "most important ones".

Heuristic Given the access graph $G(V, E, m)$:

1. associate an integer weight to each edge (see below). Construct a maximum branching $G' = (V', E', m)$ of G using the algorithm due to Edmonds,

2. for each edge in $E \setminus E'$, try to add the edge to G'. If the addition of the new edge creates a cycle of matrix weight the matrix identity or a new path with same source and destination vertices and same weight as an already existing path, the edge can be added in E'. At this step, all the communications represented by edges in G' can be made local,

3. consider the multiple paths and the cycles with $F_{p1} - F_{p2}$ or $F_{cycle} - I$ of deficient rank and try to find allocation matrices that allow to zero out even these communications.

In step 1, the weight of a branching is defined as the sum of the integer weights of the arcs in the branching. A maximum branching is any possible branching with the largest possible weight, see [8]. The simplest weight function is to assign the same value 1 to all edges of G. But we can give priority to edges according to a given criteria.

If the chosen criteria is the number of accesses due to the communication, we can associate the depth of the statement to each edge as an integer weight (we do not need the precise value, a consistent estimate is sufficient as we only want to give different priorities to the edges).

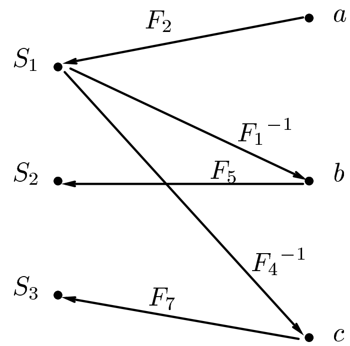

Figure 3: A possible branching

We have decided to use the rank of the access matrix as an integer weight for the following reasons: it represents the number of points of the array involved in the communication, if the access matrix is of deficient rank, the communication may be a broadcast (see Section 3).

Figure 2 represents the weighted access graph for our motivating example. A matrix weight (corresponding to the access matrix or its pseudo-inverse) and an integer weight (corresponding to the rank of the access matrix) are associated to each edge.

A possible maximum branching for our example is represented in Figure 3. It contains 5 edges out of the 7 edges of the access graph. Hence, 5 communications can be made local and 2 communications remain nonlocal communications. Note that both edges of maximum weight 3 have been zeroed out.

2.3 Optimizing residual communications

The main problem now is: what to do with the residual general communications? After the previous heuristic, the communication corresponding to the access matrix F_6 (reading a in S_1) and the one corresponding to the access matrix F_3 (reading a in S_1) remain nonlocal general communications.

To simplify the equations, we use the matrix $F'_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}$ (which satisfies the equation $GF_1 = \mathrm{Id}$) instead of $F_1{}^{-1}$ and $F'_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ (which satisfies $GF_4 = \mathrm{Id}$) instead of $F_4{}^{-1}$ (see the remark in Section 2.2.2).

Back to the branching, there is one input vertex a. Hence, we can choose the allocation matrix M_a and deduce the other allocation matrices from M_a in order to make the communications local:

$M_{S_1} = M_a F_2$, $M_b = M_{S_1} F'_1$, $M_c = M_{S_1} F'_4$, $M_{S_2} = M_b F_5$ and $M_{S_3} = M_c F_7$.

Choose, for example, $M_a = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. We have $M_{S_1} = M_a F_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $M_b = M_a F_2 F'_1 = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix}$, $M_{S_2} = M_a F_2 F'_1 F_5 = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix}$, $M_c = M_a F_2 F'_4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$, and $M_{S_3} = M_a F_2 F'_4 F_7 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$.

We check that the two communications corresponding to the two edges of the access graph that do not belong to the selected branching remain nonlocal communications:

$M_a F_6 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix} \neq M_{S_2}$ and $M_a F_3 = \begin{pmatrix} 1 & 1 \\ -4 & -5 \end{pmatrix} \neq M_{S_1}$.

Remark If we left-multiply M_a by a unimodular matrix U of $M_n(\mathbb{Z})$ ($M_n(\mathbb{Z})$ denotes the set of $n \times n$ matrices over $\mathbb{Z}$ and the unimodular matrices of $M_n(\mathbb{Z})$ are those of determinant ± 1), all the allocation matrices deduced from M_a will be left-multiplied by the same unimodular matrix. Inside each connected component of the branching, the alignment matrices are computed up to a multiplication by an unimodular matrix.

In the following we will explore the possibilities to to optimize the two residual communications.

1. **Detecting structured communications.** For the first communication concerning the access matrix F_6 ($a(F_6(i,j)^t + c_6)$ read in S_2), we can notice that F_6 has a non null kernel: $\ker F_6$ is the subset generated by the vector $v = (0,1,1)^t$.

   Let I_1 and I_2 two points of the index set of S_2 such that $I_2 - I_1 \in \ker F_6$, i.e $I_2 - I_1 = kv$, $k \in \mathbb{Z}$: we have $F_6 I_1 = F_6 I_2$. Besides, $M_{S_2} v = (-1,1)^t$, hence $M_{S_2} I_1 \neq M_{S_2} I_2$.

   The same value of array a, $a(F_6 I_1 + c_6)$ (or $a(F_6 I_2 + c_6)$), located in the memory of processor $M_a(a(F_6 I_1 + c_6)) + \alpha_a$, must be read by distinct processors $M_{S_2} I_1 + \alpha_{S_2}$ and $M_{S_2} I_2 + \alpha_{S_2} = M_{S_2}(I_1 + kv) + \alpha_{S_2}$. Hence, the communication can be viewed as a partial broadcast along one direction of the processor space.

Structured communications such as broadcasts are efficiently implemented on modern DMPCs. On a CM-5, the ratio between the communication time for a general communication and a broadcast is 60 (see Table 1). The broadcast can be total (the value is sent to the whole processor space) or partial (the value is sent in only some directions of the processor space). To be efficient a partial broadcast must be performed along directions of the processor space. To optimize the first residual communication, we choose the allocation matrices to have a partial broadcast along one axis of the processor space.

In our example, the value $a(F_6 I_1 + c_6)$ in the memory of the processor $M_a(a(F_6 I_1 + c_6)) + \alpha_a$ is sent to processors $M_{S_2}(I_1 + kv) + \alpha_{S_2}$, $k \in \mathbb{Z}$. The communication can be decomposed in a translation and a partial broadcast along the direction given by the vector $M_{S_2} v$. However,

$$M_{S_2} v = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

With such a mapping, the broadcast is not parallel to an axis. We *rotate* the mapping by left multiplying M_a (and therefore all the other allocation matrices) by a suitable unimodular matrix. Let $V = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. We have $V M_{S_2} v = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Hence, by left-multiplying all the allocations matrices by the unimodular matrix V, the communication corresponding to the access matrix F_6 becomes a macro-communication that can be performed efficiently[3].

2. **Decomposing the last residual communication.** For the second communication concerning the access matrix F_3 ($a(F_3(i,j)^t + c_3)$ read in S_1), we try to decompose it into more simple and efficient data movements such as horizontal or vertical ones.

   In our example, the processor $P = V M_a F_3 I$ sends its data to the processor $Q = V M_{S_1} I$ (up to a translation). Let T be the routing matrix: a processor P sends data to

[3] We point out that the rank-deficient communication corresponding to F_8 also becomes a broadcast parallel to one direction of the processor space: $\ker F_8$ is the subset generated by the vectors $v_1 = (0,1,-1)^t$ and $v_2 = (1,0,-1)^t$. We have $V M_{S_3}[v_1\ v_2] = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. Of course this is only a lucky coincidence!

processor $Q = TP$. We have $TVM_aF_3 = VM_{S_1}$. So, $T = VM_{S_1}(M_aF_3)^{-1}V^{-1}$. Besides

$$VM_{S_1}(M_aF_3)^{-1}V^{-1} = \begin{pmatrix} 1 & -1 \\ 5 & -4 \end{pmatrix}.$$

We can decompose $VM_{S_1}(M_aF_3)^{-1}V^{-1}$ into two elementary matrices that correspond to horizontal or vertical communications (see Section 4):

$$VM_{S_1}(M_aF_3)^{-1}V^{-1} = \begin{pmatrix} 1 & 0 \\ 5 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}.$$

To summarize, in our example, we finally obtain on the access graph 5 local communications, one broadcast and one residual communication that can be decomposed into two elementary communications.

Remark In this paper we only describe how to minimize the number of general communications and how to optimize the residual general communications. The last step of the mapping process would consist in minimizing the number of local communications (translations). The heuristic proposed by Darte and Robert in [4] in the case of uniform loop nests can for example be utilized.

3 Structured communications

In this section we derive formal conditions for detecting and implementing structured communications such as broadcasts, scatters, gathers, and reductions. We also address the message vectorization problem.

3.1 Broadcast

Broadcasts occur when the same data item is accessed at same time-step by several virtual processors. Consider the following loop nest:

Example 2

for I do

S(I) $\ldots = a(F_aI + c_a)$

endfor

Let $M_S I + \alpha_S$ and $M_a I + \alpha_a$ be the affine allocation functions for statement S and for array a. We assume that the computation time steps for $S(I)$ are given by a linear multi-dimensional schedule. Let Θ_S be the multi-dimensional scheduling application for statement S, the computation of $S(I)$ is scheduled at time-step $t = \Theta_S(I)$ (see [9]) on the processor $M_S I + \alpha_S$ and the data accessed is $a(F_a I + c_a)$ which is located in the processor $M_a(F_a I + c_a) + \alpha_a$.

The same index x from array a is read at same time-step by several processors if there exist two indices I_1, I_2 of the iteration space such that:

1. $t = \Theta_S I_1 = \Theta_S I_2$,

2. $x = F_a I_1 + c_a = F_a I_2 + c_a$,

3. $M_S I_1 + \alpha_S \neq M_S I_2 + \alpha_S$.

This implies that $I_1 - I_2 \in \ker(\Theta_S) \cap \ker(F_a) \setminus \ker(M_S)$. Let IS be the index set for statement S. Let $I_0 \in$ IS, let p the dimension of $\ker(\Theta_S) \cap \ker(F_a) \setminus \ker(M_S)$ and $(v_1, v_2, \ldots, v_p)$ vectors that generate $\ker(\Theta_S) \cap \ker(F_a) \setminus \ker(M_S)$. Let $v \in \ker(\Theta_S) \cap \ker(F_a) \setminus \ker(M_S)$, the instances $S(I_0)$ and $S(I_0 + v)$ are scheduled at the same time-step and the same element $a(F_a I_0 + c_a)$ of the array a is read. The (value of the) data item $a(F_a I_0 + c_a)$ is sent to both processors $M_S I_0 + \alpha_S$ and $M_S(I_0 + v) + \alpha_S$. Hence, the necessary communications for the set of computations $\{S(I) \mid I = I_0 + v,\ v \in \ker(\Theta_S) \cap \ker(F_a) \setminus \ker(M_S)\}$ can be regrouped into two communications: first a translation of the data item $a(F_a I_0 + c_a)$ from $M_a(F_a I_0 + c_a) + \alpha_a$ to $M_S I_0 + \alpha_S$, then a broadcast of this item along the vectors $M_S v_1, M_S v_2, \ldots, M_S v_p$.

Let m be the dimension of the target virtual processor space. If $p = m$, the broadcast is total. If $0 < p < m$, the broadcast is partial. See Figures 4 and 5 for examples with $m = 2$ and $p = 2$ or $p = 1$. If $p = 0$, the broadcast is hidden by the mapping and we have only a point to point communication. In the case of a partial broadcast, as Platonoff in [19], we impose to broadcast in directions parallel to some axes of the processor space. In Figure 5, the direction of the broadcast $M_S v_1$ is parallel to one axis.

Partial broadcast conditions Let D be the $m \times p$ matrix $\begin{bmatrix} M_S v_1 & M_S v_2 & \ldots & M_S v_p \end{bmatrix}$. In the case of partial broadcast, $0 < p < m$, D is a narrow rectangular matrix. Partial

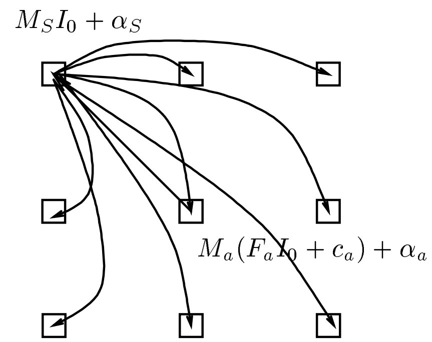

Figure 4: Complete broadcast, $m = 2$, $p = 2$

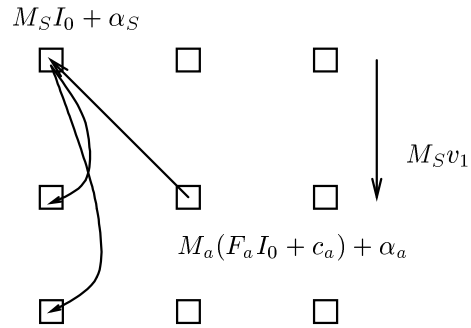

Figure 5: Partial broadcast, $m = 2$, $p = 1$

broadcasts correspond to efficient schemes of communications iff there are along some dimensions of the processor space, *i.e*, $D = \begin{bmatrix} D_1 \\ 0 \end{bmatrix}$ (up to a row permutation), where D_1 is $n \times p$ full rank matrix ($n \leq p$) and 0 is the $(m-n) \times p$ null matrix.

If D is not of the previous form, we use the right Hermite form of D to find a new allocation matrix M_S such that the broadcast is made parallel to some axis: we decompose D as $D = Q \begin{bmatrix} H \\ 0 \end{bmatrix}$ where Q is a unimodular matrix and H is $n \times n$ lower triangular matrix. Hence, $\begin{bmatrix} Q^{-1}M_Sv_1 & Q^{-1}M_Sv_2 & \ldots & Q^{-1}M_Sv_p \end{bmatrix} = \begin{bmatrix} H \\ 0 \end{bmatrix}$. If we left multiply M_S by the unimodular matrix Q^{-1}, the partial broadcast becomes parallel to the axis.

3.2 Scatter

A scatter occurs when several data located in the same processor must be sent at same time-step to several processors. The only difference between broadcast and scatter is that different data are sent to the receiving processors. Consider again Example 2, the conditions to have a scatter are the following:

1. $t = \Theta_S I_1 = \Theta_S I_2$,
2. $p = M_a(F_aI_1 + c_a) + \alpha_a = M_a(F_aI_2 + c_a) + \alpha_a$,
3. $M_SI_1 + \alpha_S \neq M_SI_2 + \alpha_S$,
4. $F_aI_1 + c_a \neq F_aI_2 + c_a$.

This implies that $I_1 - I_2 \in (\ker \Theta_S \cap \ker M_a F_a) \backslash (\ker M_S \cap ker F_a)$. We have the same conditions on partial scatters as on partial broadcasts.

3.3 Gather

A gather is the "inverse" operation of a scatter. Several data located in different processors are sent at same time-step to the same processor. Consider the following loop nest:

Example 3

for I do
    S(I) $a(F_a I + c_a) = \ldots$
endfor

Again, the conditions for a gather are close to that for scatters:

1. $t = \Theta_S I_1 = \Theta_S I_2$,
2. $p = M_a(F_a I_1 + c_a) + \alpha_a = M_a(F_a I_2 + c_a) + \alpha_a$,
3. $M_S I_1 + \alpha_S \neq M_S I_2 + \alpha_S$,
4. $F_a I_1 + c_a \neq F_a I_2 + c_a$.

This implies that $I_1 - I_2 \in (\ker \Theta_S \cap \ker M_a F_a) \backslash (\ker M_S \cap ker F_a)$. A gather can be partial or total and we have the same conditions on partial gathers as previously.

3.4 Reduction

A reduction is similar to a gather but the different values sent to the same processor are used to compute one single value. Reductions occur when, at same time-step, a single processor uses values computed by different instances of the same instruction on different processors. A reduction is usually associated with a commutative and associative function $(+, \min, \ldots)$ that computes a resulting value from many input values. Consider the following loop nest, where s represents an array element:

Example 4

for I do

S(I) $s = s + b(F_b I + c_b)$

endfor

The conditions to have a reduction are:

1. $t = \Theta_S I_1 = \Theta_S I_2$,

2. $M_b(F_b I_1 + c_b) + \alpha_b \neq M_b(F_b I_2 + c_b) + \alpha_b$,

3. $M_S I_1 = M_S I_2$.

This implies that $I_1 - I_2 \in \ker \Theta_S \cap \ker M_S \backslash \ker M_b F_b$.

3.5 Message vectorization

Message vectorization can take place when a processor accesses data from another processor that remains unchanged for several consecutive time steps: data items to be sent can be grouped into packets that are sent just in time to reach their destination. The communications can be extracted out of a loop and performed before the computations. The idea is to replace a set of small-size communications by a single large message, so as to reduce overhead due to start-up and latency [11, 21, 5].

Consider again Example 2. The space-time transformation is given by:

$$\begin{pmatrix} t \\ p \end{pmatrix} = \begin{pmatrix} \Theta_S \\ M_S \end{pmatrix} I + \begin{pmatrix} 0 \\ \alpha_S \end{pmatrix},$$

where p is the processor responsible at time-step t (the schedule can be multi-dimensional) for the computation $S(I)$. Let $\tilde{S} = \begin{pmatrix} \Theta_S \\ M_S \end{pmatrix}$. Hence, we have $I = \tilde{S}^{-1} \begin{pmatrix} t \\ p - \alpha_S \end{pmatrix}$. For the computation $S(I)$, processor p reads data from processor $M_a(F_a I + c_a) + \alpha_a$. This expression does not depend on t when $M_a F_a \tilde{S}^{-1} = (0, X)$ where 0 is a $m \times p$ null matrix and X is a $m \times m$ matrix. This condition is equivalent to $M_a F_a = (0, X)\tilde{S} = X M_S$ and therefore to $\ker M_S \subset \ker(M_a F_a)$.

4 Communication decomposition

In this section we explain how to decompose general affine communications into elementary (horizontal or vertical) ones. We provide analytical conditions for such a decomposition to exist. In Section 4.1 we report experimental communication times on the Intel Paragon that demonstrate the usefulness of communication decomposition.

4.1 Communication decomposition: why

Consider a general affine communication occurring for statement $S(I) : ... = ...a(F_a I + c_a)$. Using notations of Section 2, the virtual processor $p_{recv} = M_S.I + \alpha_S$ (where M_S is of size $m \times d$) receives a data item from the virtual processor $p_{send} = M_a.(F_a.I + c_a) + \alpha_a$ (where M_a is of size $m \times q_a$). Assume for simplicity that M_S, M_a and F_a are nonsingular square matrices of size $m \times m$ (hence $m = d = q_a$). We have the following equations:

$$M_a^{-1}(p_{\text{send}} - \alpha_a) = F_a I + c_a$$

$$F_a^{-1}[M_a^{-1}(p_{\text{send}} - \alpha_a) - c_a] = I$$

$$p_{\text{recv}} = M_S F_a^{-1}[M_a^{-1}(p_{\text{send}} - \alpha_a) - c_a] + \alpha_S$$

Hence $p_{\text{recv}} = M_S F_a^{-1} M_a^{-1} p_{\text{send}} + \text{const}$. Let $T = M_S F_a^{-1} M_a^{-1} = M_s (M_a.F_a)^{-1}$ be the routing matrix.

The idea is to decompose T into the product of several elementary matrices that will generate communications parallel to one axis of the virtual processor space. We mainly discuss the case where the determinant of T is equal to 1: $\det T = 1$. Assume $m = 2$ for example: we aim at decomposing T into the product of matrices $L_i = \begin{pmatrix} 1 & 0 \\ l_i & 1 \end{pmatrix}$ (horizontal communications) or $U_i = \begin{pmatrix} 1 & k_i \\ 0 & 1 \end{pmatrix}$ (vertical communications). Some current-generation machines have a 2D-topology (Intel Paragon) or 3D-topology (Cray T3D), hence the case $m = 2$ and $m = 3$ are of particular practical interest. Due to space limitation we detail only the case $m = 2$ hereafter, but the ideas can be obviously extended to higher dimensions.

To give a concrete motivation, consider the following routing matrix $T = \begin{pmatrix} 1 & 3 \\ 2 & 7 \end{pmatrix}$. The communication is to be implemented on a Paragon machine configured as a 3×8 grid of processors. Table 2 reports communication ratios when implementing T directly, or when decomposing it as $T = LU$, where $L = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$ and $U = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}$.

Communication	Not decomposed	L	U	$L.U$
Execution ratio	10	1.5	3.9	5.4

Table 2: Decomposing versus not decomposing an affine communication on the Intel Paragon

Table 2 shows that decomposing the communication gives better results (intuitively, better have several simple communications than a complicated one). The cost for U is higher than for L because of the larger grid dimension. Data was distributed using a standard cyclic distribution. Let us notice that the gain obtained by decomposing communication is machine- and compiler-dependent. Table 2 is only intended to give experimental evidence that communication decomposition can prove efficient. To be conservative, we look to decomposing general communications into a small (say $l \leq 4$) number of elementary ones.

4.2 Communication decomposition: how

4.2.1 Direct decomposition

Let $T = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $\det(T) = ad - bc = 1$. We derive conditions to know whether T can be decomposed into the product of less than or equal to four elementary matrices L_i and U_i.

The necessary and sufficient condition to decompose the matrix T in the product of two elementary matrices is $d = 1$ or $a = 1$. In the same way, the necessary and sufficient condition for T to be equal to the product of three elementary matrices is: c divides $a - 1$ or b divides $d - 1$, and the necessary and sufficient condition for a matrix to be equal to the product of four elementary matrices is: $\exists \alpha, b - \alpha.d | d - 1$ or $\exists \alpha, c - \alpha.a | a - 1$ (see [6]).

We could go further and look for a decomposition into five or more elementary matrices. But an exhaustive search shows that every 2×2 matrix T with $\det(T) = 1$ and whose coefficients are all lower than or equal to 14 in absolute value, is equal to the product of 2, 3 or 4 elementary matrices. In practice, larger coefficients are unlikely to be encountered in loop nests !

4.2.2 With left-multiplication by a unimodular matrix

As outlined in Section 2.3, alignment matrices are computed up to a multiplication by an unimodular matrix. If we left-multiply M_S and M_a by a unimodular matrix M, then the routing matrix $T = M_s(M_a.F_a)^{-1}$ is transformed into MTM^{-1}. Therefore, rather than decomposing T into the product of elementary matrices, we can search for a unimodular matrix M such that MTM^{-1}, a matrix *similar* to T, can be decomposed into such a product.

Consider again the case $m = 2$ and $\det(T) = 1$. The best would be to show that T is similar to a product LU of two elementary matrices, so as to decompose MTM^{-1} into one horizontal communication followed by one vertical communication. The problem amounts to show that every integer matrix $T = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $ad - bc = 1$ is similar over $\mathbb{Z}$ to a matrix $LU = \begin{pmatrix} 1 & \alpha \\ \beta & 1+\alpha\beta \end{pmatrix}$. The problem is surprisingly difficult, and the answer depends upon the coefficients of T.

However, a sufficient condition to have a matrix T similar to the product of two elementary matrices is the same as the necessary and sufficient condition for T to be decomposed into the product of three elementary matrices (see [6]). Either strategy could be more interesting, depending upon the target machine. Note that all integer matrices T with $\det(T) = 1$ and whose coefficients are all lower than or equal to 5 are similar to a product of 2 elementary matrices.

We have only dealt with routing matrices of determinant 1. In [6], we briefly explain how to generalize to arbitrary matrices. Besides, we also introduce in [6] a new data distribution scheme called *grouped partition*. This distribution is well-suited to implementing horizontal/vertical communications on the Paragon and leads to smaller communication times than the standard `CYCLIC` or `CYCLIC(BLOCK)` distributions.

5 Summary

Summarizing previous sections, we can sketch our complete heuristic.

1. **Zero out non local communications**

(a) **Access graph** Construct the access graph $G = (V, E, m)$ and extract a maximum branching $G' = (V', E', m)$ from G.

(b) **Multiple paths, cycles**

i. For each edge in $E \setminus E'$, try to add the edge to G'. If the addition of the new edge creates a cycle of matrix weight the matrix identity or a new path with same source and destination vertices and same weight as an already existing path, the edge can be added in E'. At this step, all the communications represented by edges in G' can be made local.

ii. In each connected component, consider the multiple paths and the cycles with $F_{p_1} - F_{p_2}$ or $F_{cycle} - I$ of deficient rank and try to find allocation matrices that allow to zero out even these communications.

2. **Optimize residual communications** For each connected component of the graph obtained after step 1:

(a) **Structured communications** Detect the possible macro-communications and compute the conditions on allocation matrices to have efficient communications after mapping on the virtual processor space. If the conditions are not satisfied, it is possible in each connected component to left-multiply all the allocation matrices by a unimodular matrix (see Section 3).

(b) **Decompose residual general communications** Decompose the residual general communications in more simple and efficient ones. If the allocation matrices are not yet fixed in a connected component, to obtain a better decomposition, again left-multiply the allocation matrices by a unimodular matrix (see Section 4).

6 Evaluation of the heuristic

We have tested the heuristic on classical examples from signal processing or linear algebra, see [12]. The chosen algorithms are `Gauss`, `Cholesky`, `Seidel` (system resolution), `matrix multiplication`, `Lanczos` (eigenvalue computation), and `Burg's method` (signal processing). For each algorithm, we give the number of resulting communications for dimensions

Program	Edge nb	Dim	Communication type				
			Null	Local	Broadcast	Decomposable	General
Gauss	7	1	5	0	2 (rank 1)	0	0
		2	4	0	3 (rank 1)	0	0
Cholesky	9	1	6	0	2	0	1
		2	4	0	3 (rank 1)	0	2
Seidel	7	1	5	0	2	0	0
		2	5	0	2 (partial)	0	0
Matrix multiplication	5	1	4	0	1	0	0
		2	3	0	2 (rank 1)	0	0
Lanczos	45	2	22	7	12 (rank 1)	1	3
Burg	21	2	12	0	8 (rank 1)	0	1

Table 3: Results after residual communication optimization

of the target architecture which seem to be of interest (generally 1 or 2 for the chosen algorithms). The results are gathered in Table 3.

The two-step mapping heuristic gives very interesting results. The optimization process takes off most of the general communications that remain after the first step. These results must of course be completed by actual experiments onto DMPCs. They are theoretical and they only constitute a first step in the evaluation process.

7 Related work

7.1 Literature overview

The data alignment problem has motivated a vast amount of research. A brief review of some related work is presented here. The review contains only short abstracts of the presented papers and is by no mean intended to be comprehensive.

Knobe, Lukas and Steele [14] discuss techniques for automatic layouts of arrays in a compiler targeted to SIMD architectures. The approach to data locality is to consider each occurrence of a datum as a separately allocated object and to mark preferences among certain occurrences to indicate that they should be allocated together. This approach is extended in [18] to MIMD systems. In [17], Lukas shows that same data optimization alignment techniques can be used in both distributed and shared memory systems.

Huang and Sadayappan [13] consider the issue of communication-free hyperplane partitioning. By modeling the iteration and data spaces and the relation that maps one to another, necessary and sufficient conditions for the feasibility of communication-free partitioning along hyperplanes are characterized.

Li and Chen [16] formulate the problem of *index domain alignment* as finding suitable alignment functions that embed the index domains of the arrays into a common index domain. The problem of minimizing the cost of data movement is shown to be NP-complete and a greedy heuristic is proposed.

Anderson and Lam [1], propose a communication-free algorithm that determines alignment for both data and computations. The algorithm is based on the mathematical model of decompositions as affine functions and is structured into three components: partition, orientation and displacement. The only parallelism exploited is *forall* parallelism or *doacross* parallelism using tiling.

Bau *et al.* [2] propose a simple linear framework to solve the communication-free alignment problem.

Darte and Robert [4, 5] introduce a *communication graph* that contains all the information to align data and computations. They prove that, even in the simple case of uniform perfect nested loops, the problem is NP-complete and they give heuristics.

Feautrier [10] proposes a greedy algorithm analogous to Gaussian elimination to determine a placement function. Data and computations are aligned in such a way that the *owner computes rule* is respected.

Platonoff propose an heuristic based on the greedy heuristic given by Feautrier [10], enlarged by a detection of structured communications such as broadcasts, whose cost can be an order of magnitude smaller than for a general communication (see Table 1). Platonoff's algorithm is divided on 4 steps: (i) a prototype mapping function is written for each instruction; (ii) broadcasts are located in the initial code. To find them, Platonoff uses the *data flow graph*, defined by Feautrier; (iii) broadcasts are processed. The broadcast directions are identified, and conditions on the prototype mapping functions are written to preserved the structured communication (the projection onto the virtual processor space must not be along the broadcast directions). For partial broadcasts, conditions are written to have broadcasts parallel to some axes of the processor space; (iv) the volume of residual communications is minimized with the greedy heuristic given in [10].

7.2 Discussion

Many authors have proposed heuristics to find a communication-free mapping or to the minimize the number of communications. NP-completeness results show that the problem is difficult [16, 4, 7, 1].

The strategies described in [1] and [2] are very similar, only the frameworks are different. They both propose a communication-free algorithm which will generally lead to a trivial mapping with all computations and data grouped on the same processor. With the heuristic given by Dion and Robert [7], the dimension of the target architecture is an input of the mapping process and data and computations are mapped on the whole processor space. Feautrier [10] takes into account the dimension of the scheduling: the rank of the mapping function is equal to the depth of the statement minus the dimension of the scheduling.

Besides, in [10], the *owner computes rule* is respected whereas other authors [1, 7] have relaxed it.

We compare our heuristic with the strategy developed by Platonoff on a small example. Consider the following loop nest:

Example 5

for $t = 1, n$ *do*
    for $i, j, k = 1, n$ *do*
        $S(I) : a(t, i, j, k) = b(t, i, j)$
endfor

Let $\{\vec{e_i}\}_{i=1,4}$ be the canonical basis of the iteration space. We assume here that the loop nest is scheduled by a linear scheduling vector $\pi = \vec{e_1}$. The outer loop is sequential and the inner loops on i, j ,and k are parallel. The arrays a and b can be accessed before and after the parallel loop, inside the sequential loop. We also assume that we want to map computations and data onto a 2-dimensional processor space: $m = 2$. The access matrix for array b is

$F_b = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. A necessary condition to have a broadcast is $\ker(\pi^t) \cap \ker(F_b) \neq 0$. In our example, we have $\ker(\pi^t) \cap \ker(F_b) = \vec{e_4}$. According to the allocation matrices, the broadcast can be kept or masked. Platonoff's strategy consists in:

1. detecting all the macro-communications possibly present in the initial program,
2. writing the conditions to preserve these macro-communications,
3. making local (or zeroing out) as many residual communications as possible, by taking into account the conditions previously found in his prototype mapping function.

To keep the partial broadcast, Platonoff would choose $M_S = M_a = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ and $M_b = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. With this mapping, each processor sends a value to a column of processors. At each time-step, n^2 broadcasts of one element along one dimension of the processor space are necessary. The loop nest is computable with n^3 macro-communications.

However, we easily see that, with our strategy, the loop nest can be computed without any communication! For example, we can choose $M_b = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $M_S = M_a = M_b F_b = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. We first try to make local as many communications as possible and then we try to extract macro-communications from the residual communications, whereas Platonoff first detects the macro-communications and then try to zero out the residual communications.

Let us compare Platonoff's results with the results given in Table 3 for `Burg` algorithm:

	Null	Local	Broadcasts	General
Platonoff's heuristic	32%	14%	14%	40%
Results in Table 3	57%	0%	38%	5%

When detecting macro-communications first, some other communications remain general whereas they could have been made local.

8 Conclusion

Many authors have proposed heuristics to minimize the communication volume or number when mapping data and computations of an affine loop nests onto DMPCs. It is generally impossible to obtain a communication-free mapping and another goal in the mapping process is to "optimize" in some sense the residual communications.

We have designed an efficient two-step heuristic

1. based upon the access graph to zero out as many communications as possible, with priority given to communications of largest volume

2. enlarged with the processing of residual communications, either through the extraction of structured communications or through the decomposition of complex communications into simpler ones

We have provided a detailed analysis of structured communications (broadcasts, scatters, gathers, reductions) and of message vectorization, together with criteria for their efficient mapping. We have also given analytical formulae to decompose complex communications, and we have shown that such a decomposition improves communication performance on the Paragon.

We have evaluated our mapping strategy on classical examples from linear algebra or signal processing. Most of the communications can be made local, and for the residual general communications it is possible to take advantage of the structured communication facilities usually implanted onto DMPCs. For general patterns of communication like broadcasts or reductions, the situation is clear: they correspond to very efficient communications and it is worthwhile to use them. For communication decomposition (decomposition of a general communication into a sequence of simple communications), it is efficient but more tests have to be performed on DMPCs to evaluate more precisely the set of general communications that can be efficiently decomposed. Of course, what we have presented is a "theoretical" evaluation of the mapping heuristic. It could be also interesting to actually test the efficiency of the produced code on DMPCs.

Acknowledgment

We would like to thank the referees for their comments and suggestions.

References

[1] Jennifer M. Anderson and Monica S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices*, 28(6):112–125, June 1993.

[2] David Bau, Induprakas Kodukula, Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill. Solving alignment using elementary linear algebra. In K. Pingali, U.Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and compilers for parallel computing - 7th international workshop*, volume LNCS 892, pages 46–60. Springer -Verlag, 1994.

[3] Tzung-Shi Chen and Jang-Ping Sheu. Communication-free data allocation techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):924–938, 1994.

[4] Alain Darte and Yves Robert. Mapping uniform loop nests onto distributed memory architectures. *Parallel Computing*, 20:679–710, 1994.

[5] Alain Darte and Yves Robert. On the alignment problem. *Parallel Processing Letters*, 4(3):259–270, 1994.

[6] Michèle Dion, Cyril Randriamaro, and Yves Robert. How to optimize residual communications ? Research Report 95-27, Ecole Normale Supérieure de Lyon, October 1995.

[7] Michèle Dion and Yves Robert. Mapping affine loop nests: New results. In Bob Hertzberger and Guiseppe Serazzi, editors, *High-Performance Computing and Networking, International Conference and Exhibition*, volume LCNS 919, pages 184–189. Springer-Verlag, 1995. Extended version available as Technical Report 94-30, LIP, ENS Lyon (anonymous ftp to lip.ens-lyon.fr).

[8] J.R. Evans and E. Minieka. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker Inc, 1992.

[9] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II, multidimensional time. *Int. J. Parallel Programming*, 21(6):389–420, December 1992.

[10] Paul Feautrier. Towards automatic distribution. *Parallel Processing Letters*, 4(3):233–244, 1994.

[11] Hans Michael Gerndt. *Automatic parallelization for distributed-memory multiprocessing systems.* PhD thesis, University of Bonn, 1989.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins, 2 edition, 1989.

[13] C.H. Huang and P. Sadayappan. Communication-free hyperplane partitioning of nested loops. In Banerjee, Gelernter, Nicolau, and Padua, editors, *Languages and Compilers for Parallel Computing*, volume 589 of *Lecture Notes in Computer Science*, pages 186–200. Springer Verlag, 1991.

[14] Kathleen Knobe, Joan D. Lukas, and Guy L. Steele. Data optimization: Allocation of arrays to reduce communication on SIMD machines. *Journal of Parallel and Distributed Computing*, 8:102–118, 1990.

[15] A. Korganoff and M. Pavel-Parvu. *Eléments de théorie des matrices carrées et rectangles en analyse numérique.* Dunod, 1966. (In French).

[16] Jingke Li and Marina Chen. The data alignment phase in compiling programs for distributed memory machines. *Journal of Parallel and Distributed Computing*, 13:213–221, 1991.

[17] Joan D. Lukas. Data locality for shared memory. In *Sixth SIAM Conf. Parallel Processing for Scientific Computing*, volume 2, pages 836–839. SIAM Press, 1993.

[18] Joan D. Lukas and Kathleen Knobe. Data optimization and its effect on communication costs in MIMD fortran code. In Dongarra, Kennedy, Messina, Sorensen, and Voigt, editors, *Fifth SIAM Conf. Parallel Processing for Scientific Computing*, pages 478–483. SIAM Press, 1992.

[19] Alexis Platonoff. *Contribution à la Distribution Automatique des Données pour Machines Massivement Parallèles*. PhD thesis, École Nationale Supérieure des Mines de Paris, March 1995.

[20] W. Shang and Z. Shu. Data alignment of loop nests without nonlocal communications. In *Application Specific Array Processors*, pages 439–450. IEEE Computer Society Press, 1994.

[21] Chau-Wen Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines*. PhD thesis, Rice University, 1993.

[22] Michael Wolfe. The Tiny loop restructuring research tool. In H.D. Schwetman, editor, *International Conference on Parallel Processing*, volume II, pages 46–53. CRC Press, 1991.