How to optimize residual communications?

Michèle Dion, Cyril Randriamaro and Yves Robert

Laboratoire LIP, URA CNRS 1398
Ecole Normale Supérieure de Lyon, F - 69364 LYON Cedex 07
e-mail: [mdion,crandria,yrobert]@lip.ens-lyon.fr

Abstract

Minimizing communications when mapping affine loop nests onto distributed memory parallel computers has already drawn a lot of attention. This paper focuses on the next step: as it is generally impossible to obtain a communication-free (or local) mapping, how to optimize the residual communications? We explain how to take advantage of macro-communications such as broadcasts, scatters, gathers or reductions or how to decompose general affine communications into simpler ones that can be performed more efficiently. We finally give a two-step heuristic that summarizes our approach: first minimize the number of nonlocal communications, then optimize residual affine communications using macro-communications or decompositions.

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.

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 [11, 7, 9, 1, 2, 8, 4].

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 non-local 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 communications. A natural question arises: is there a way to “optimize” in some sense the communications that remain?

Platonoff [12] 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 macro-communications such as a broadcast or a reduction. Also, translations are much more efficient than general (affine) communications.

<table>
<thead>
<tr>
<th>Reduction</th>
<th>Broadcast</th>
<th>Translation</th>
<th>General comm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
<td>2.5</td>
<td>90</td>
</tr>
</tbody>
</table>

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

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 communication parallel to the grid axes (horizontal and vertical) proves much faster, as illustrated in Section 4.

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

1. Zero out as many non-local communications as possible. To this purpose, we use an heuristic modified from Dion and Robert [4]. This heuristic is based upon the access graph and will be explained in Section 2. We weight the edges of the access graph to take communication volume into account.

2. For remaining 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 macro-communication (in addition to reduction and broadcast, we could have a scatter or a gather, for instance).

   (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 and 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 macro-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. Finally, concluding remarks are stated in Section 6. Some technical results are not detailed in this short version; the reader can refer to [3] for an extended version.

2 Motivating example

In this section, we informally explain our new approach on an example. First we apply an heuristic modified from [4] to minimize the number of nonlocal communications. Then, we try to make the residual nonlocal communications “efficient”.

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 $N$
  for $j = 0$ to $M$
    \[
    \begin{align*}
    \{S_1\} & b(F_1(i, j)^t + c_1) = g_1(a(F_2(i, j)^t + c_2), \\
    & a(F_3(i, j)^t + c_3), c(F_4(i, j)^t + c_4)) \end{align*}
    \]
    for $k = 0$ to $N + M$
      \[
      \begin{align*}
      \{S_2\} & b(F_5(i, j, k)^t + c_5) = g_2(a(F_6(i, j, k)^t + c_6)) \\
      \{S_3\} & c(F_7(i, j, k)^t + c_7) = g_3(a(F_8(i, j, k)^t + c_8)) \end{align*}
      \]
    endfor
  endfor
endfor

where

\[
F_1 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & -1 \end{pmatrix} \quad c_1 = \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix}, \quad F_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad c_2 = \begin{pmatrix} -1 \\ 1 \\ -6 \end{pmatrix}, \quad F_3 = \begin{pmatrix} 1 & 1 \\ -4 & -5 \end{pmatrix}
\]

and $c_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, $F_4 = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$ and $c_4 = \begin{pmatrix} -1 \\ 0 \\ 3 \end{pmatrix}$, $F_5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I_{d_3}$ and $c_5 = \begin{pmatrix} 2 \\ 5 \\ -3 \end{pmatrix}$, $F_6 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$ and $c_6 = \begin{pmatrix} 1 \\ 5 \\ 2 \end{pmatrix}$.

$F_7 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$ and $c_7 = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$, $F_8 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and $c_8 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

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 [14] for instance) can be used, for example, to perform dependence analysis, all loops are DOALL loops, hence all computations can be executed at the same time-step.

Mapping the loop nest onto a $d$-dimensional virtual processor space consists in determining allocation matrices for each statement and for each array. Affine allocation functions are defined as in [1, 4]:

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 $d \times d$ matrix and $\alpha_S$ is a $d$-vector.
2. for each array $x$ of dimension $q_x$: alloc$_a(I) = M_x I + \alpha_x$, where $M_x$ is a $m \times q_x$ matrix and $\alpha_x$ 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 $M_S I + \alpha_x$. For instance, in statement $S_1$, the value $a(F_2(i,j) + c_2)$ has to be read in the memory of the processor

$$\text{alloc}_{S_1}((i,j)^t) = M_{S_1}(F_2(i,j) + c_2) I + \alpha_{S_1}$$

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

$$\text{alloc}_{S_1}((i,j)^t) = M_{S_1}(F_2(i,j) + c_2) I + \alpha_{S_1}.$$ 

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

$$\delta_{a,S_1} = \text{alloc}_{S_1}((i,j)^t) - \text{alloc}_{a}(F_2(i,j) + c_2)$$

Then, another objective of the mapping process, as recognized by [1], is to zero out the nonlocal part of the communication. 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 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 statement $S$ of depth $d$ and array $x$ of dimension $q_x$: the equation is $M_S = M_x F$, where $M_S$ is an $m \times d$ matrix, and $M_x$ a $m \times q_x$ matrix. The matrix $F$ is of dimension $q_x \times d$ and we consider only full rank access matrices. 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 [3]), 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 $F^{-1}$ of size $d \times q_x$ and of rank $q_x$, such that $FF^{\top} = Id$ (see [3]). Hence, if there exists $M_x$ such that $M_S = M_x F$, then $M_x F^{-1} = M_S F^{-1} = M_x$. 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 $m$.

---

$^1$We search for full rank allocation matrices: otherwise the target $m$-dimensional processor space would not be fully utilized.

$^2$Compatibility condition: $M_S = M_S F^{-1} F$. It ensures that the mapping is a valid mapping from the virtual processor to the physical processor space.
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 [3]). 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.

2.2.2 Access graph

We recall here the definition of a $m$-dimensional access graph $G = (V,E,m)$ as stated in [4]. 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 where 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 made local, the weight of the edge is $F$; and if $d < q_x$ 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 made local, the 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 $F_3$ is not represented because $F_3$ 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 $GF = Id$. Any matrix $H$ such that $H = F^{-1} + M(\text{Id} - FF^{-1})$ where $M$ is an arbitrary matrix of correct dimension also satisfies $GF = Id$. Hence, in the access graph, we can choose any full rank matrix $G$ such that $GF = Id$ as weight matrix (and not necessarily the “true” left pseudo-inverse $F^{-1}$ as defined in [10]).

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 $a \rightarrow S_1 \rightarrow b \rightarrow S_2$, 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 $F_0$ are made local: we successively let $M_{S_1} = M_a F_2, M_b = M_{S_1} F_1^{-1}$ and $M_{S_2} = M_b F_0$.

There is another path from $a$ to $S_2$ with the direct edge $a \rightarrow S_2$ (access matrix $F_0$ for reading $a$ in $S_2$). Is it possible to make local this communication in addition to the three above communications? Using the edge $a \rightarrow S_2$ we get the equation $M_{S_2} = M_a F_0$ while we had $M_{S_2} = M_a F_2 F_1^{-1} F_5$, with $F_5 = M_a F_2 F_1^{-1} F_5$.

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 $GF_1 = Id$) 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 $F_2 F_1 F_5 = F_6$. 

![Figure 1. Access graph ($m = 1$ or $m = 2$)](image1.png)

![Figure 2. Access graph with integer weights](image2.png)
In our example, let \( F_{pah} = F_2F_1F_3 \). We compute \( F_{pah} = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \neq F_6 \).

Note that as \( F_{pah} - F_6 = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \) is of deficient rank, according to \( m \), it will or not be possible to find a matrix \( M_a \) such that the condition \( M_aF_2F_1^{-1}F_4 = M_aF_6 \) 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 [4] 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 \( \text{Id} \) 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”. We use the following heuristic:

**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 \( F_{p_1} - F_{p_2} \) or \( F_{cycle} - I \) of deficient rank and try to find allocation matrices that allow to zero out even these communications.

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 [5]. The simplest weight function is to assign the same value 1 to all edges of \( G \). But we can give priority to edges that involve a large volume of communication. We do not need to precisely know the volume of data exchanged, a consistent estimate is sufficient. Consider the following loop nest:

\[
\text{for } i \text{ do } \{ \text{Statement } S \} a(F_aI + c_a) = \ldots b(F_bI + c_b) \ldots \text{ endfor}
\]

We propose to use the dimension of the set \( \{ J \mid \exists I, J = F_aI + c_a \} \) (resp. \( \{ J \mid \exists I, J = F_bI + c_b \} \)) to estimate the volume of data exchanged for the communication corresponding to the access matrix \( F_a \) (resp. \( F_b \)). Hence, in the access graph, an integer weight corresponding to the rank of the access matrix is associated to each edge. In this way, communications inducing the largest traffic are zeroed out in priority. Note that in the heuristic proposed by Feautrier [7], the volume of data exchanged induced by a communication is taken into account in a similar way.

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 volume of data exchange due to the communication) are associated to each edge. 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 \( S_1 \)) too 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 \( GF_1 = \text{Id} \)) instead of \( F_1^{-1} \) and \( F'_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \) (which satisfies \( GF_2 = \text{Id} \)) instead of \( F_2^{-1} \) (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:

\[
M_{S_1} = M_a F_2, \quad M_b = M_{S_1} F'_1, \quad M_c = M_{S_1} F'_4, \quad M_{S_2} = M_b F_3 \quad \text{and} \quad M_{S_3} = M_c F_7.
\]

Choose, for example, \( M_a = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \). We have

\[
M_{S_1} = M_a F_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad M_b = M_a F_2 F'_1 = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix}, \quad M_{S_2} = M_a F_2 F'_4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{and} \quad M_{S_3} = M_a F_2 F'_4 F_7 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.
\]

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:

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

**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 \( \pm 1 \)), 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.

**How to optimize the two residual communications?**

1. **Detecting macro-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 \( M_a(a(F_6 I_1 + c_6)) + a \), must be read by distinct processors \( M_{S_2} I_1 + \alpha \).

   and \( M_{S_2} I_2 + \alpha = M_{S_2}(I_1 + kv) + \alpha \). Hence, the communication can be viewed as a partial broadcast along one direction of the processor space.

   Macro-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 in section 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 most efficient a partial broadcast must be performed along the directions of the processor space. To optimize the first residual communications, we choose the allocation matrices so as 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 \( M_a(a(F_6 I_1 + c_6)) + a \), is sent to processors \( M_{S_2}(I_1 + kv) + \alpha \), \( k \in \mathbb{Z} \). The communication can be decomposed in a translation and a partial broadcast along the direction given by the vector \( M_{S_2} v \). However,

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

   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.

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 data flow matrix: a processor \( P \) sends data to processor \( Q = TP \). We have \( TV M_{S_1} F_3 = V M_{S_1} \). So,

   \[
   TV = VM_{S_1}(aF_3)^{-1} V^{-1}.
   \]

   Besides

   \[
   VM_{S_1}(aF_3)^{-1} V^{-1} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \]

---

\(^{3}\)We point out that the rank-deficient communication corresponding to \( F_6 \) also becomes a broadcast parallel to one direction of the processor space: \( \ker F_6 \) is the subset generated by the vectors \( v_1 = (0, 1, -1)^t \) and \( v_2 = (1, 0, -1)^t \). We have \( VM_{S_2} v_1 \) \( v_2 \) = \( \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \). Of course this is a lucky coincidence!
We can decompose $V M S_i (M_a F_3) V^{-1}$ into two elementary matrices that correspond to horizontal or vertical communications (see Section 4):

$$VM S_i (M_a F_3) V^{-1} = \left(\begin{array}{cc}
1 & 0 \\
5 & 1 \\
\end{array}\right) \left(\begin{array}{cc}
1 & -1 \\
0 & 1 \\
\end{array}\right).$$

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.

### 3 Macro-communications

In this Section we derive formal conditions for detecting and implementing macro-communications such as broadcasts, scatters, gathers, and reductions.

#### 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

\[ S(I) \ldots = a(F_a I + c_a) \]

don

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 $\Theta_S$ be the multi-dimensional scheduling application for statement $S$, the computation of $S(I)$ is scheduled at time-step $t = \Theta_S(I)$ (see [6]) 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 $S(I_0+v)$ 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 [12], we impose to broadcast in directions parallel to some axis 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 $[M_S v_1 \ M_S v_2 \ \ldots \ M_S v_p]$. In the case of partial broadcast, $0 < p < m$, $D$ is a narrow rectangular matrix. Partial broadcasts correspond to efficient schemes of communications if 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 a $(m-n) \times p$ 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 \( D = Q \begin{bmatrix} H & 0 \\ \hline 0 & 0 \end{bmatrix} \) 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 is to be 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. \( p = M_a(F_a I_1 + c_a) + \alpha_a = M_a(F_a I_2 + c_a) + \alpha_a \),
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 \( I_1 - I_2 \in (\ker \Theta_S \cap \ker M_a F_a) \setminus (\ker M_S \cap \ker F_a) \). We have similar conditions for a scatter as for a broadcast.

### 3.3 Gather

A gather is the “inverse” operation to a scatter. Several data located in different processor 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 \( I_1 - I_2 \in (\ker \Theta_S \cap \ker M_a F_a) \setminus (\ker M_S \cap \ker F_a) \). A gather can be partial or total and we have similar 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. \( M_b(F_b I_1 + c_b) + \alpha_b = M_b(F_b I_2 + c_b) + \alpha_b \),
3. \( M_S I_1 = M_S I_2 \).

This implies that \( I_1 - I_2 \in \ker \Theta_S \cap \ker M_b \setminus \ker M_b F_b \).

### 4 Communication decomposition

Consider a general affine communication occurring for statement \( S(I) : \ldots = a(F_a I + c_a) \). 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:

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

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

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

Hence \( p_{recv} = M_S F_a^{-1} M_a^{-1} p_{send} + \text{const} \). Let \( T = M_S F_a^{-1} M_a^{-1} \) be the data-flow 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: \( \det T = 1 \) (see [3] for extensions). Assume \( m = 2 \) for example: we aim at decomposing \( T \) into the product of matrices \( L_i = \begin{bmatrix} 1 & 0 \\ li & 1 \end{bmatrix} \) or \( U_i = \begin{bmatrix} 1 & k_i \\ 0 & 1 \end{bmatrix} \) (horizontal or
vertical communications). We would have similar elementary matrices for larger dimensions: an elementary matrix would look like

$$L_i = \begin{pmatrix} 1 & 0 \\ 0 & \cdots \\ \alpha_0.k & \cdots & 1 & \cdots & \alpha_n.k \\ \cdots & 0 \\ 0 & 1 \end{pmatrix}$$

(and similarly for $U_i$). 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.

In [3], we give necessary and sufficient conditions in the case $m = 2$ to decompose the communications into three or four elementary communications. Furthermore, an exhaustive search have shown that every $2 \times 2$ 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 practise, larger coefficients are unlikely to be encountered in loop nests!

Besides, by left-multiplying the allocation matrices by a suitable unimodular matrix, it is possible to reduce the length of the sequence of elementary communications.

To give a concrete motivation, consider the following data-flow matrix $T = \begin{pmatrix} 1 & 3 \\ 2 & 7 \end{pmatrix}$. The communication is to be implemented on a Paragon machine configured as a $3 \times 8$ 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 $U = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}$.

<table>
<thead>
<tr>
<th>Communication</th>
<th>Not decomposed</th>
<th>$L$</th>
<th>$U$</th>
<th>$LU$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution ratio</td>
<td>10</td>
<td>1.5</td>
<td>3.9</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Table 2. Decomposing versus not decomposing a general 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.

Anyway, 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.

In [3], we also introduce a new data distribution scheme that enables to implement elementary communications more efficiently.

## 5 Summary

Summarizing previous Sections, we can sketch our complete heuristic.

### 1. Zero out non local communications

1. **Access graph** Construct the access graph $G = (V, E, m)$ associated to the loop nest.

2. **Branching** Extract a maximum branching $G' = (V', E', m)$ from the access graph.

3. **Multiple paths, cycles**
   
   (a) 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.

   (b) 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:

1. **Macro communications** Detect the possible macrocommunications 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).

2. **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 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,
2. enlarged with the processing of residual communications, either through the extraction of macro-communications of through the decomposition of complex communications into simpler ones.

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

In the first step of the heuristic, we only consider the volume of data exchange to determine which communications are zeroed out in priority. To give a better cost assignment, we could privilege macro-communications and communication decompositions, and take into account the characteristics of the target machine.

Besides, at the end of the mapping process, an heuristic analogous to the one described in [2] in the case of uniform loop nests can be apply to zero out local communications.

References


