An Array Partitioning Analysis for Parallel Loop Distribution

Marc Le Fur, Jean-Louis Pazat and Françoise André *

IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, FRANCE

Abstract. This paper presents the compilation techniques implemented in a compiler for a HPF-like language. The stress is especially put on the description of an optimized scheme which is dedicated to the compilation of parallel nested loops. The generation of the SPMD code is based on the polyhedral model and allows for the partitioning of the arrays involved in the loop in order to achieve symbolic restriction of iteration domains and message aggregation. Experimental results for some well-known kernels are shown.

1 Introduction and Motivation

The data parallel model is often shown as a promising way to easily write programs for distributed memory computers or clusters of workstations.

Among data parallel languages, High Performance Fortran [12] and its precursors embed data partitioning features in a sequential language as a means to drive the parallelization and the distribution of programs. With this paradigm, the programmer is still provided with a familiar uniform logical address space and a sequential flow of control. The compiler generates code according to the SPMD model and the links between the code execution and the data distribution are enforced by the *owner-computes rule*: each processor executes only the statements that modify the data assigned to it by the user-specified distribution.

This approach constitutes the basis of several compilers [17, 6] and is also applied in the PANDORE environment [4].

A simple scheme called *runtime resolution* [5] permits the translation of any sequential program into communicating processes. The first experiments have shown that aggressive optimization techniques are needed to generate efficient code.

The paper presents an optimized translation scheme, implemented in the PANDORE compiler, dedicated to the compilation of parallel loop nests with one statement. The importance of parallel loop nests in scientific applications is manifest. On the one hand, these loops form most computation-intensive parts of scientific programs. On the other hand, these loops can be produced thanks to automatic parallelization techniques such as affine-by-statement scheduling [10, 9] or automatic vectorization [1].

* mlefur@irisa.fr, pazat@irisa.fr, fandre@irisa.fr

We address the problem of determining the communication and computation sets induced by the user-supplied distribution of arrays. The work presented in the paper is related to [2, 18], [17] and [6] that use respectively polyhedrons, regular sections and overlaps (rectangular sections) to represent these sets. The problem has also been studied in the framework of the compilation of array statements in [7] and [11]; this time, a finite state machine and triplets permit the characterization of the different sets.

The paper is organized as follows. We first give the principles of the optimized compilation scheme in section 2 and then illustrate the technique in section 3. Notations and definitions are posed in section 4 in order to present code generation in 5. Experimental results for well-known kernels are shown in section 6.

2 Principles of the Optimized Scheme

The optimized compilation scheme separates the generated SPMD code into a communication part and a computation part. The generation of each part relies on a domain analysis that takes advantage of the user-partitioning of the arrays into rectangular blocks in order to achieve symbolic restriction of iteration domains and message aggregation. Furthermore, this domain analysis is symbolic, *i.e.* independent of the number of blocks of each array involved in the parallel loop nest.

Our optimized compilation method applies to parallel loop nests where array references and loop bounds are affine functions of the enclosing indices and where each loop stride is equal to one. As regards data distribution, we assume that an array is either replicated on all processors (the array is owned by each processor) or distributed, that is partitioned into rectangular blocks with constant sizes (known at compile-time), each block being assigned to exactly one processor. Scalar elements are systematically replicated.

Because we assume that the loop bounds and the array access functions are affine, the data access domains can be characterized by polyhedrons and the generated code execution will consist in scanning these polyhedrons. Indeed, a loop nest performing the enumeration of its integer vectors can be associated with any non empty bounded polyhedron. Different algorithms can be used to solve the polyhedron scanning problem [13, 8, 14]; the algorithm implemented in the PANDORE compiler is detailed in [15].

3 Example

For the sake of concreteness, let us consider the following (contrived) parallel nest:

for $i_1 = 1\ ,\ 1000$
  for $i_2 = i_1\ ,\ 2 * i_1 + 1$
    $S(i_1, i_2):\ X[i_1, i_2 - i_1] := Y[i_2, 2 * i_1 - 2]$

where arrays X and Y are partitioned into 8 blocks numbered from 0 to 7, as indicated in figure 1.

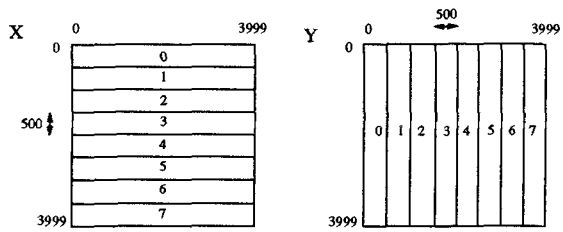

Fig. 1. Partitioning of Arrays

Communication Code Generation.

a - The set of vectors $(j^X\ j^Y\ i_1\ i_2)$ where j^X (resp. j^Y) $\in 0..7$ is a block of X (resp. Y), $(i_1\ i_2)$ an iteration vector such that $S(i_1, i_2)$ writes in j^X and reads in j^Y, can be characterized by the polyhedron $\mathcal{P}_1$ defined by the system:

$$\begin{cases} 0 \leq j^X \leq 7 \\ 0 \leq j^Y \leq 7 \\ 1 \leq i_1 \leq 1000 \\ i_1 \leq i_2 \leq 2 * i_1 + 1 \\ 500 * j^X \leq i_1 \leq 500 * j^X + 499 \\ 500 * j^Y \leq 2 * i_1 - 2 \leq 500 * j^Y + 499 \end{cases}$$

b - The enumeration code of polyhedron $\mathcal{P}_1$ can be computed by one of the algorithms [15, 13, 8, 14]; for instance, the algorithm implemented in PANDORE [15] yields the following nested loop:

```
for jX = 0 , 2
   for jY = max(0 , 2*jX-1) , min(3 , 2*jX+1)
      for i1 = max(250*jY+1 , 500*jX) ,
               min(250*jY+250 , 500*jX+499)
         for i2 = i1 , 2*i1+1
```

It is important to notice that the first two loops of this enumeration code do not scan the whole cartesian product 0..7 x 0..7, as it can be seen in figure 2. The subset of 0..7 x 0..7 described by the (j^X, j^Y)-loop is defined as the convex-hull of the integer projection of polyhedron $\mathcal{P}_1$ along the i_1, i_2 axes.

c - We then insert two masks and a communication instruction in this enumeration code in order to generate the SPMD send code and then a dual SPMD receive code. The send code is generated as follows:

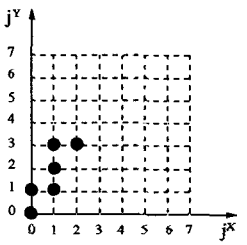

Fig. 2. Restriction of Iteration Domains

```
for j^X = 0 , 2
   if myself ≠ owner of block j^X of X then
      for j^Y = max(0 , 2*j^X −1) , min(3 , 2*j^X +1)
         if myself = owner of block j^Y of Y then
            for i1 = max(250*j^Y +1 , 500*j^X) ,
                     min(250*j^Y +250 , 500*j^X +499)
               for i2 = i1 , 2*i1 +1
                  pack Y[i2, 2 * i1 − 2] in buffer
            send buffer to the owner of block j^X of X
```

The runtime library routine *pack* performs data elements aggregation. Although the previous loop contains masks, the reader should note that these masks are evaluated at the block level and not at the iteration vector level as in the runtime resolution. Furthermore, the (j^X, j^Y)-loop enumerates only a few vectors, as seen in figure 2, and the location of the first mask prevents from enumerating all these vectors.

Computation Code Generation.

a - As before, the set of vectors $(j^X\ i_1\ i_2)$ where $j^X \in 0..7$ is a block of X, $(i_1\ i_2)$ an iteration vector such that $S(i_1, i_2)$ writes in j^X, can be characterized by the polyhedron $\mathcal{P}_2$ defined by the following system:

$$\begin{cases} 0 \leq j^X \leq 7 \\ 1 \leq i_1 \leq 1000 \\ i_1 \leq i_2 \leq 2 * i_1 + 1 \\ 500 * j^X \leq i_1 \leq 500 * j^X + 499 \end{cases}$$

b - The vectors of $\mathcal{P}_2$ can be enumerated by the nested loop:

```
for j^X = 0 , 2
   for i1 = max(500*j^X, 1) , min(500*j^X +499, 1000)
      for i2 = i1 , 2*i1 +1
```

which shows that the blocks 3..7 of X are not written during the computation (the j^X-loop scans the convex-hull of the integer projection of polyhedron $\mathcal{P}_2$ along the i_1, i_2 axes).

c - Finally, a mask is inserted to produce the SPMD computation code:

```
for j^X = 0 , 2
   if myself = owner of block j^X of X then
      for i1 = max(500*j^X, 1) , min(500*j^X +499, 1000)
         for i2 = i1 , 2*i1+1
            X[i1, i2 - i1] := Y[i2, 2 * i1 - 2]
```

4 Notations and Definitions

If x is a row or column vector with n components, x_q $(1 \leq q \leq n)$ stands for the q^{th} component of x. Given a row or column vector u with n components $u_1, \ldots, u_n$, $X[u]$ denotes the reference $X[u_1, \ldots, u_n]$ to array X. If the access function associated with an array reference is affine, for instance $X[i+3, 2i+j+1]$, the reference may be noted in matrix form as follows:

$$X[\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 3 \\ 1 \end{pmatrix}]$$

Finally, for row vectors u and v with n and p components respectively, $(u\ v)$ stands for the vector with $(n+p)$ components $(u_1 \ldots u_n\ v_1 \ldots v_p)$. This notation can be extended to an arbitrary number of vectors.

Notations related to Distributed Arrays. In order to simplify the notations, we assume that the lower bound is 0 in each dimension of an array. Let X be a m-dimensional distributed array and let h_p^X (resp. s_p^X) be the number of elements (resp. the block size) of array X in the p^{th} dimension $(p \in 1..m)$.

We note $Part(X) = \{p \in 1..m \,/\, s_p^X < h_p^X\}$ the set of partitioned dimensions of X and $d(X,q)$ the q^{th} partitioned dimension of X if $q \in 1..|Part(X)|$. For arrays Y and Z given in figure 3 for instance:

$h_1^Y = 400$ $h_2^Y = 800$	$h_1^Z = 500$ $h_2^Z = 700$
$s_1^Y = 200$ $s_2^Y = 250$	$s_1^Z = 500$ $s_2^Z = 300$
$Part(Y) = \{1, 2\}$	$Part(Z) = \{2\}$
$d(Y,1) = 1$ $d(Y,2) = 2$	$d(Z,1) = 2$

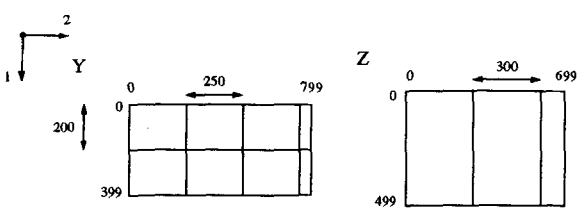

Fig. 3. Partitioning of Arrays

Let n be the number of partitioned dimensions of array X. We term block of X indexed by the vector j, where j is a row vector with n components such that $\forall q \in 1..n \quad 0 \leq j_q \leq \lceil h^X_{d(X,q)}/s^X_{d(X,q)} \rceil - 1$, and we note $Block(X, j)$, the block $X[lbnd_1..ubnd_1, \ldots, lbnd_m..ubnd_m]$ of array X defined by

$lbnd_{d(X,q)} = j_q s^X_{d(X,q)}$ and $ubnd_{d(X,q)} = min(j_q s^X_{d(X,q)} + s^X_{d(X,q)} - 1, h^X_{d(X,q)} - 1)$ for each partitioned dimension $d(X,q)$ of X $(q \in 1..n)$,
$lbnd_p = 0$ and $ubnd_p = h^X_p - 1$ for each dimension p of X which is not partitioned.

Intuitively, the components of j are related to the coordinates of $Block(X, j)$ in the space of the blocks of array X when all the dimensions of X are partitioned. In the general case, j must be related only to the coordinates of $Block(X, j)$ associated with the partitioned dimensions of X. For arrays Y and Z for instance:

$Block(Y, (0\ 1)) = Y[0..199, 250..499]$, $Block(Y, (1\ 3)) = Y[200..399, 750..799]$.
$Block(Z, (0)) = Z[0..499, 0..299]$, $Block(Z, (2)) = Z[0..499, 600..699]$.

and more generally:

$Block(Y, (j_1\ j_2)) = Y[200j_1..200j_1 + 199, 250j_2..min(250j_2 + 249, 799)] \quad \forall j_1 \in 0..1 \quad \forall j_2 \in 0..3$,
$Block(Z, (j_1)) = Z[0..499, 300j_1..min(300j_1 + 299, 699)] \quad \forall j_1 \in 0..2$.

Finally, for a vector u with m components and a vector j with n components, we note $Belong(X, u, j)$ the set of inequalities that must be satisfied by vector u so that the reference $X[u]$ belongs to the block $Block(X, j)$ of X:

$j_q s^X_{d(X,q)} \leq u_{d(X,q)} \leq j_q s^X_{d(X,q)} + s^X_{d(X,q)} - 1$ for each $q \in 1..n$,
$u_p \leq \lceil h^X_p / s^X_p \rceil - 1$ for each partitioned dimension p of X such that h^X_p is not a multiple of s^X_p.

Indeed, if p is a partitioned dimension of X, the constraints $u_p \leq \lceil h^X_p / s^X_p \rceil - 1$ where h^X_p is a multiple of s^X_p are implicit in this system and thus useless. For instance, for the vector $u = (i_1 + 1 \quad i_1 + 2i_2)$:

$Belong(Y, u, (j_1\ j_2)) = \{200j_1 \leq i_1 + 1 \leq 200j_1 + 199, \quad 250j_2 \leq i_1 + 2i_2 \leq 250j_2 + 249, \quad i_1 + 2i_2 \leq 799\}$,
$Belong(Z, u, (j_1)) = \{300j_1 \leq i_1 + 2i_2 \leq 300j_1 + 299, \quad i_1 + 2i_2 \leq 699\}$.

Notations related to Nested Loops. In the following, a perfectly nested loop whose (row) iteration vector is i, iteration domain $\mathcal{D}$ and body B will be noted

for i **in** $\mathcal{D}$ / B or **for** $i : Ai^T + b \geq 0$ / B

if the iteration domain of the nested loop is a polyhedron defined by the set of affine constraints $Ai^T + b \geq 0$.

Polyhedrons for Code Generation. Let us consider a parallel loop nest whose iteration vector is i and whose iteration domain is defined by the system of affine constraints $Ai^T + b \geq 0$. The generation of the communication and computation codes for the loop nest lies in the synthesis of polyhedrons called $\mathcal{P}_1$ and $\mathcal{P}_2$ which are functions of the references to distributed arrays appearing in the nest assignment.

Given two references $X[C^X i^T + d^X]$ and $X'[C^{X'} i^T + d^{X'}]$ to distributed arrays in the parallel nest assignment, $\mathcal{P}_1(X[C^X i^T + d^X], X'[C^{X'} i^T + d^{X'}])$ is the set of (row) vectors of the form $((j_p^X)_{p \in 1..|Part(X)|} \ (j_q^{X'})_{q \in 1..|Part(X')|} \ i)$ and satisfying the system of inequalities:

$$\forall p \in 1..|Part(X)| \quad 0 \leq j_p^X \leq \lceil h^X_{d(X,p)} / s^X_{d(X,p)} \rceil - 1$$
$$\forall q \in 1..|Part(X')| \quad 0 \leq j_q^{X'} \leq \lceil h^{X'}_{d(X',q)} / s^{X'}_{d(X',q)} \rceil - 1$$
$$Ai^T + b \geq 0$$
$$Belong(X, \ C^X i^T + d^X, \ (j_p^X)_{p \in 1..|Part(X)|})$$
$$Belong(X', \ C^{X'} i^T + d^{X'}, \ (j_q^{X'})_{q \in 1..|Part(X')|})$$

One can easily check that this system defines a polyhedron because all its constraints are affine (the references to arrays X and X' are affine). In other terms, $\mathcal{P}_1(X[C^X i^T + d^X], X'[C^{X'} i^T + d^{X'}])$ is the set of vectors $((j_p^X)_{p \in 1..|Part(X)|} \ (j_q^{X'})_{q \in 1..|Part(X')|} \ i)$ such that the references $X[C^X i^T + d^X]$ and $X'[C^{X'} i^T + d^{X'}]$ belong to $Block(X, (j_p^X)_{p \in 1..|Part(X)|})$ and $Block(X', (j_q^{X'})_{q \in 1..|Part(X')|})$ respectively.

For the references $Y[i_1 + 1, i_1 + 2i_2]$ and $Z[i_2 - i_1, 3i_1 - 2]$ to the arrays shown in figure 3, located in a parallel nest whose iteration domain is defined by $\{1 \leq i_1 \leq 230, \ \ i_1 + 1 \leq i_2 \leq 350\}$, the constraints satisfied by the vectors $(j_1^Y \ j_2^Y \ j_1^Z \ i_1 \ i_2)$ of $\mathcal{P}_1(Y[i_1 + 1, i_1 + 2i_2], Z[i_2 - i_1, 3i_1 - 2])$ are the following:

$$0 \leq j_1^Y \leq 1, \ \ 0 \leq j_2^Y \leq 3$$
$$0 \leq j_1^Z \leq 2$$
$$1 \leq i_1 \leq 230, \ \ i_1 + 1 \leq i_2 \leq 350$$
$$200 j_1^Y \leq i_1 + 1 \leq 200 j_1^Y + 199, \ \ 250 j_2^Y \leq i_1 + 2i_2 \leq 250 j_2^Y + 249, \ \ i_1 + 2i_2 \leq 799$$
$$300 j_1^Z \leq 3i_1 - 2 \leq 300 j_1^Z + 299, \ \ 3i_1 - 2 \leq 699$$

For a reference $X[Ci^T + d]$ in the parallel nest, the polyhedron $\mathcal{P}_2(X[Ci^T + d])$ denotes the set of (row) vectors of the form $((j_p)_{p \in 1..|Part(X)|} \ i)$ satisfying the set of affine inequalities:

$$\forall p \in 1..|Part(X)| \quad 0 \leq j_p \leq \lceil h^X_{d(X,p)} / s^X_{d(X,p)} \rceil - 1$$
$$Ai^T + b \geq 0$$
$$Belong(X, \ Ci^T + d, \ (j_p)_{p \in 1..|Part(X)|})$$

More simply, each vector $((j_p)_{p \in 1..|Part(X)|} \ i)$ of $\mathcal{P}_2(X[Ci^T + d])$ defines an iteration vector i such that the reference $X[Ci^T + d]$ belongs to the block $Block(X, (j_p)_{p \in 1..|Part(X)|})$ of X.

With the same iteration domain as previously, $\mathcal{P}_2(Y[i_1 + 1, i_1 + 2i_2])$ is the set of vectors $(j_1 \ j_2 \ i_1 \ i_2)$ satisfying:

$$0 \leq j_1 \leq 1, \ \ 0 \leq j_2 \leq 3$$
$$1 \leq i_1 \leq 230, \ \ i_1 + 1 \leq i_2 \leq 350$$
$$200 j_1 \leq i_1 + 1 \leq 200 j_1 + 199, \ \ 250 j_2 \leq i_1 + 2i_2 \leq 250 j_2 + 249, \ \ i_1 + 2i_2 \leq 799$$

and each vector $(j_1 \ i_1 \ i_2)$ of $\mathcal{P}_2(Z[i_2 - i_1, 3i_1 - 2])$ is such that:

$$0 \leq j_1 \leq 2$$
$$1 \leq i_1 \leq 230, \ \ i_1 + 1 \leq i_2 \leq 350$$
$$300 j_1 \leq 3i_1 - 2 \leq 300 j_1 + 299, \ \ 3i_1 - 2 \leq 699$$

5 Code Generation

Actually, two compilation schemes are defined depending on whether the left hand side (*lhs*) of the assignment refers to a distributed array or a replicated array.

5.1 Compilation Scheme if the lhs refers to a Distributed Array

In this case, the parallel loop nest is of the form

for $i : Ai^T + b \geq 0$
$X[C^X i^T + d^X] := Exp\ (\mathcal{D}ist \uplus \mathcal{R}epl)$

where the array X referenced in the *lhs* is a distributed array, $\mathcal{D}ist$ the set of references to distributed arrays in the expression Exp and $\mathcal{R}epl$ the set of references to replicated variables (arrays or scalar variables) in Exp.

Communication Code Generation. Let us note $\mathcal{C}om = \mathcal{D}ist - \{X[C^X i^T + d^X]\}$ the set of references in $\mathcal{D}ist$ that *may* generate communications between processors (it is clear indeed that, if $X[C^X i^T + d^X]$ belongs to $\mathcal{D}ist$, this reference does not lead to any interprocessor communication). It should be highlighted that the set $\mathcal{C}om$ can be reduced still further if some references in $\mathcal{D}ist$ are aligned (in a HPF-manner) with $X[C^X i^T + d^X]$. In the following code for instance:

```
!HPF$ ALIGN X(K,L) WITH Y(L+1,K)
  DO I = 0, N-1
     DO J = 0, N-1
        X(2*I,2*J) = X(2*I,2*J) + Y(2*J+1,2*I) * Z(I+J)
     END DO
  END DO
```

the set $\mathcal{C}om$ is only composed of the reference `Z(I+J)`. Actually, the communication code generated by the compiler is a sequence of communication codes, one code being produced for each reference $Y[C^Y i^T + d^Y]$ in $\mathcal{C}om$ as follows:

a - Compute the code enumerating the vectors $(j^X\ j^Y\ i)$ of polyhedron $\mathcal{P}_1(X[C^X i^T + d^X], Y[C^Y i^T + d^Y])$ by one of the algorithms [15, 13, 8, 14]. This yields the nested loop:

for j^X **in** $\mathcal{D}_1$
for j^Y **in** $\mathcal{D}_2(j^X)$
for i **in** $\mathcal{D}_3(j^X, j^Y)$

where $\mathcal{D}_1$, $\mathcal{D}_2(j^X)$ and $\mathcal{D}_3(j^X, j^Y)$ denote the iterations domains associated with the iteration vectors j^X, j^Y and i respectively.

b - Insert two masks and communication instructions in this nested loop to produce the SPMD send code:

```
for j^X in D_1
    if myself ≠ owner of Block(X, j^X) then
        for j^Y in D_2(j^X)
            if myself = owner of Block(Y, j^Y) then
                for i in D_3(j^X, j^Y)
                    pack Y[C^Y i^T + d^Y] in buffer
                send buffer to the owner of Block(X, j^X)
```

c - Produce the dual SPMD receive code:

```
for j^X in D_1
    if myself = owner of Block(X, j^X) then
        for j^Y in D_2(j^X)
            if myself ≠ owner of Block(Y, j^Y) then
                receive buffer from the owner of Block(Y, j^Y)
                for i in D_3(j^X, j^Y)
                    unpack Y[C^Y i^T + d^Y] from buffer
```

The runtime library routine *unpack* extracts data elements from the buffer and copies them in the local memory of the processor.

Computation Code Generation. The computation code is produced depending only on the *lhs* reference $X[C^X i^T + d^X]$.

a - Compute the enumeration code of polyhedron $\mathcal{P}_2(X[C^X i^T + d^X])$:

```
for j^X in D_4
    for i in D_5(j^X)
```

In this loop, $\mathcal{D}_4$ and $\mathcal{D}_5(j^X)$ stand for the iteration domains associated with j^X and i respectively.

b - Produce the SPMD computation code by inserting an adequate mask:

```
for j^X in D_4
    if myself = owner of Block(X, j^X) then
        for i in D_5(j^X)
            X[C^X i^T + d^X] := Exp (Dist ⊎ Repl)
```

5.2 Compilation Scheme if the lhs refers to a Replicated Variable

The communication code generated by the compiler can be simplified and also optimized, by taking advantage of collective communication routines, when the *lhs* of the parallel loop nest:

```
for i : Ai^T + b ≥ 0
    r := Exp (Dist ⊎ Repl)
```

refers to a variable (array or scalar variable) which is replicated in the local memories.

Communication Code Generation. One communication code is produced for each $Y[C^Y i^T + d^Y]$ in $\mathcal{D}ist$ as follows:

a - Compute the enumeration code of polyhedron $\mathcal{P}_2(Y[C^Y i^T + d^Y])$:

for j^Y **in** $\mathcal{D}_1$
  for i **in** $\mathcal{D}_2(j^Y)$

b - Produce the SPMD send code:

for j^Y **in** $\mathcal{D}_1$
  if *myself* = owner of $Block(Y, j^Y)$ **then**
    for i **in** $\mathcal{D}_2(j^Y)$
      pack $Y[C^Y i^T + d^Y]$ in *buffer*
    broadcast *buffer*

c - Produce the dual SPMD receive code:

for j^Y **in** $\mathcal{D}_1$
  if *myself* $\neq$ owner of $Block(Y, j^Y)$ **then**
    receive *buffer*
    for i **in** $\mathcal{D}_2(j^Y)$
      unpack $Y[C^Y i^T + d^Y]$ from *buffer*

Computation Code Generation. According to the owner-computes rule, the parallel loop nest is replicated on all the processors:

for $i : Ai^T + b \geq 0$
  $r := Exp\ (\mathcal{D}ist \biguplus \mathcal{R}epl)$

5.3 Compiling Parameterized Loops

The method previously presented allows the compilation of *parameterized* parallel loop nests, that is to say parallel loop nests depending on variables (not assigned in the loop nest) or surrounding loop counters. Let us note k the vector of parameters associated with the loop nest and $Mk^T + h \geq 0$ the system of constraints, that may be empty, satisfied by k. In this case, the parameterized loop nest is of the form:

for $i : Ai^T + Bk^T + c \geq 0$
  $lhs\_ref := Exp\ (\mathcal{D}ist \biguplus \mathcal{R}epl)$

where $Ai^T + Bk^T + c \geq 0$ defines the iteration domain of the loop parameterized by the vector k. In the following code for instance, the inner i-loop is a parallel loop nest parameterized by the surrounding loop counter k:

for $k = 1\ ,\ 100$
  ...
  for $i = 1\ ,\ k$
    $A[i + k] := B[i] + C[2 * k + i]$
  ...

For these parameterized loops, the SPMD code is generated the same way, the polyhedrons $\mathcal{P}_1$ and $\mathcal{P}_2$ defined in section 4 being now parameterized by k.

Given two references $X[C^X i^T + D^X k^T + e^X]$ and $X'[C^{X'} i^T + D^{X'} + e^{X'}]$ to distributed arrays in the parallel nest assignment, $\mathcal{P}_1(X[C^X i^T + D^X k^T + e^X], X'[C^{X'} i^T + D^{X'} k^T + e^{X'}])$ is the set of vectors $((j_p^X)_{p \in 1..|Part(X)|}\ (j_q^{X'})_{q \in 1..|Part(X')|}\ i)$ satisfying the system of inequalities:

$$\forall p \in 1..|Part(X)| \quad 0 \leq j_p^X \leq \lceil h^X_{d(X,p)} / s^X_{d(X,p)} \rceil - 1$$
$$\forall q \in 1..|Part(X')| \quad 0 \leq j_q^{X'} \leq \lceil h^{X'}_{d(X',q)} / s^{X'}_{d(X',q)} \rceil - 1$$
$$Ai^T + Bk^T + c \geq 0$$
$$Belong(X,\ C^X i^T + D^X k^T + e^X,\ (j_p^X)_{p \in 1..|Part(X)|})$$
$$Belong(X',\ C^{X'} i^T + D^{X'} k^T + e^{X'},\ (j_q^{X'})_{q \in 1..|Part(X')|})$$

For a reference $X[Ci^T + Dk^T + e]$ in the parallel nest, the polyhedron $\mathcal{P}_2(X[Ci^T + Dk^T + e])$ denotes the set of vectors $((j_p)_{p \in 1..|Part(X)|}\ i)$ satisfying the set of inequalities:

$$\forall p \in 1..|Part(X)| \quad 0 \leq j_p \leq \lceil h^X_{d(X,p)} / s^X_{d(X,p)} \rceil - 1$$
$$Ai^T + Bk^T + c \geq 0$$
$$Belong(X,\ Ci^T + Dk^T + e,\ (j_p)_{p \in 1..|Part(X)|})$$

Again, the algorithms [15, 13, 8, 14] permit the generation of a nested loop scanning these parameterized polyhedrons in the context $Mk^T + h \geq 0$.

6 Experiments

6.1 Runtime Support

The runtime resolution and the optimized scheme rely on the implementation of a paged array management [16] which tries to balance the speed of accesses and the memory requirements. The runtime library routine *pack* used in the optimized scheme performs several communication optimizations. Direct communication is performed whenever possible; what is transferred in this case is a memory zone that is contiguous both on the sender and the receiver, thus eliminating any need of coding/decoding between message buffers and local memories. Message aggregation is also carried out and reduces the effect of latency by grouping small messages into a large message. Furthermore, the routine *pack* eliminates redundant communications that may occur with non injective access functions or when several references to the same distributed array appear in the right hand side.

Because of these optimizations (messages exchanged between processors are generally composed of contiguous memory zones), it should be noted that the compiler does not exactly produce the receive codes given in 5.1 and 5.2; the array eléments are not element-wise unpacked from the received buffer.

6.2 Experimental Results

Some results of experiments with the optimized compilation scheme are presented in this section. Performance results are shown in figure 4 for two kernels: Cholesky factorization and Jacobi relaxation; the description of the parallelization of a wave propagation application can be found in [3].

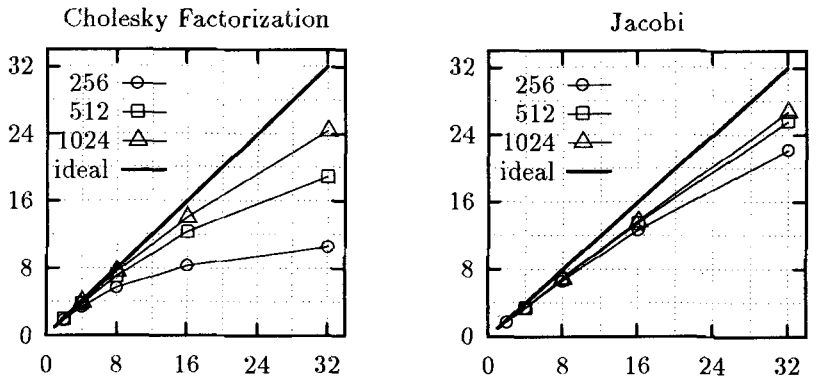

Fig. 4. Speedup

Measurements have been performed on a 32-node iPSC/2. The presented graphs show the speedup against the number processors for several input sizes. Speedup is defined as the parallel time over the time of the original sequential program measured on one node. The obtained efficiencies are satisfactory, ranging from 85% to 95% on 8 processors and reaching around 80% on 32 processors for the largest data size.

7 Conclusion

In this paper, we have presented an optimized compilation technique for parallel loop nests expressed in HPF-like languages. This scheme has been fully implemented in the PANDORE compiler and cohabits with the runtime resolution, thus permitting the compilation of the whole input language. The optimized method performs a symbolic polyhedron-based domain analysis that exploits the partitioning of the arrays involved in the computation in order to achieve restriction of iteration domains and message aggregation. The scope of this scheme can be extended to more general regular loops by integrating parallelization techniques that produce automatically the parallel loops that can be handled by our technique.

The performances obtained on a series of numerical kernels are satisfactory even though enhancements can be made along several axes. First, we plan to

improve the compilation technique in order to avoid the multiple enumerations of the same memory location that may occur with non injective access functions or when several references to the same array appear in the right hand side. At the moment, the runtime support prevents from translating these multiple enumerations into multiple sends. This problem can be handled at compile-time by scanning directly the polyhedron affine image, or at least a superset, associated with a right hand side array reference. Moreover, this scanning can be reorganized to exploit the contiguity in the local representation of the distributed arrays in order to maximize *direct communication*, that is the communication of contiguous memory zones.

References

1. R. Allen and K. Kennedy. Automatic Translation of Fortran Programs to Vector Form. *ACM TOPLAS*, 9(4), October 1987.
2. C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell. A Linear Algebra Framework for Static HPF Code Distribution. In *Fourth International Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, December 1993.
3. F. André, M. Le Fur, Y. Mahéo, and J.-L. Pazat. Parallelization of a Wave Propagation Application using a Data Parallel Compiler. In *Nineth International Parallel Processing Symposium*, Santa Barbara, California, April 1995.
4. F. André, M. Le Fur, Y. Mahéo, and J.-L. Pazat. The Pandore Data-Parallel Compiler and its Portable Runtime. In *High-Performance Computing and Networking*, LNCS 919, Springer Verlag, Milan, Italy, May 1995.
5. D. Callahan and K. Kennedy. Compiling Programs for Dis tributed-Memory Multiprocessors. *Journal of Supercomputing*, 2, 1988.
6. B.M. Chapman and H.P. Zima. *Compiling for Distributed-Memory Systems.* Research Report ACPC/TR 92-17, Austrian Center for Parallel Computation, University of Vienna, November 1992.
7. S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H Teng. Generating Local Addresses and Communication Sets for Data-Parallel Programs. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, California, May 1993.
8. J.-F. Collard, P. Feautrier, and T. Risset. *Construction of DO Loops from Systems of Affine Constraints.* Research Report 93-15, LIP, Lyon, France, May 1993.
9. A. Darte and Y. Robert. Constructive Methods for Scheduling Uniform Loop Nests. *IEEE Transactions on Parallel and Distributed Systems*, 5(8), August 1994.
10. P. Feautrier. Some Efficient Solutions to the Affine Scheduling Problem, Part I, One-Dimensional Time. *International Journal of Parallel Programming*, 21(5), 1992.
11. S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan. *Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines.* Technical Report 19, The Ohio State University, 1994.
12. High Performance Fortran Forum. *High Performance Fortran Language Specification.* Technical Report Version 1.0, Rice University, May 1993.
13. F. Irigoin and C. Ancourt. Scanning Polyhedra with DO Loops. In *Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 39–50, April 1991.

14. H. Le Verge, V. Van Dongen, and D. K. Wilde. *Loop Nest Synthesis Using the Polyhedral Library.* Research Report 2288, INRIA, France, May 1994.
15. M. Le Fur. Scanning Parameterized Polyhedron using Fourier-Motzkin Elimination. In *High Performance Computing Symposium*, Montréal, Canada, July 1995.
16. Y. Mahéo and J.-L. Pazat. Distributed Array Management for HPF Compilers. In *High Performance Computing Symposium*, Montréal, Canada, July 1995.
17. C.W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines.* PhD thesis, Rice University, January 1993.
18. V. Van Dongen. Compiling Distributed Loops onto SPMD Code. *Parallel Processing Letter*, 4(3), 1994.