ICPS
Informatique et
Calcul Parallèle
de Strasbourg

Publication 94-09

Data Compiling for Systems of Affine Recurrence Equations

C. Mongenet

*Published in ASAP'94, August 22-24, 1994, San Francisco (USA),
p. 212-223, IEEE Computer Society Press*

ICPS - Université Louis Pasteur
Pôle API, Boulevard Sébastien Brant, F-67400 Illkirch

Data Compiling for Systems of Affine Recurrence Equations

Catherine Mongenet

Université Louis Pasteur, Département d'Informatique
7 rue René Descartes, 67084 Strasbourg France
Email: mongenet@dpt-info.u-strasbg.fr

Abstract

In order to get a parallel solution from a system of affine recurrence equations, a space-time transformation must first be determined. Such a transformation is characterized by a schedule and an allocation. In the context of data parallelism, efficient compilers require among other criteria appropriate data compiling techniques. These techniques should take into account the communication primitives of the target architecture (in particular broadcast primitives if any) and minimize the number of communications. The objective of this paper is to deal with these two questions (efficient use of broadcast capabilities and communication minimization) in order to determine efficient space-time transformations from systems of affine recurrence equations.
Keywords: *parallelism, mapping, systems of recurrence equations, data parallelism, data compilation, communications.*

1: Introduction

Data parallelism is often considered as a general programming model in the context of massively parallel architectures ([7]). This model is founded on the virtualization of data and processes. A program is composed of parallel operations in which the mechanisms managing the data accesses are hidden. To be efficiently used this programming model requires powerful tools such as parallelizing compilers. Such a compiler must optimize the use of the different communication primitives available on the target architecture. Two types of primitives are usually distinguished : the broadcast primitives and the local neighbor-to-neighbor communications. One of the key issues of these compilers is the alignment: how to allocate the elementary computations to the processors. An efficient use of the target machine requires the alignment to minimize the communications.

The different works in this domain find their foundations both in the systolic synthesis ([9], [11], [6], [2], [16], [17], [13]) and in the loop parallelization techniques ([12], [10], [21], [5], [19], [20], [4] [8], [18], [1] and the enclosed references). In both studies one deals with problems expressed by systems of recurrence equations or equivalent loop nests. A parallel solution is obtained by applying a space-time transformation to the problem. This transformation is characterized by a schedule or timing function (usually affine) and by an allocation function (usually linear). The class of equations tackled nowadays are the systems of affine recurrence equations (SARE in the following). These systems extend the systems of uniform recurrence equations (SURE) characterized by uniform dependencies. The affine dependencies yield multiple-uses of some data. One may deal with these multiple-uses by broadcasting. The use of broadcast primitives results in a more optimal affine schedule, i.e. a schedule which optimizes the global computation time of the algorithm.

When a target architecture offers broadcast primitives one should try to maximize the broadcasts in order to optimize the schedule and get a fast algorithm. However, in the context of transformation with affine schedules and linear allocations, it is usually not possible to apply broadcast to all the dependencies of a problem. Therefore to get a fast and efficient solution, it is important to minimize the communications, i.e. to localize the data associated with the remaining dependencies as much as possible.

The aim of this paper is to present the different aspects of the dependency compiling from systems of affine recurrence equations. We focus on dependency compiling for result variables. Compiling of data variables induces alignment of data arrays and has been widely studied in the context of loop parallelization. This paper is an extension of a previous work restricted to the systems of uniform recurrence equations [15]. It is organized as follows. The main results on systems of affine recurrence equations, dependency analysis and space-time mapping are presented in section 2. Section 3 discusses how to compile a result dependency with broadcast primitives while section 4 describes how to compile it with local primitives only. We then present in section 5 how to use these two approaches to determine an efficient space-time transformation associated with a problem and how to generate the corresponding parallel code. The results are illustrated by the LL^t Cholesky factorization in section 6.

2: Systems of recurrence equations and automatic parallelization

A system of affine recurrence equations is composed of equations of the form :

$$X[z] = f(\ldots, Y[g(z)], \ldots), \qquad z \in D_p, \tag{E}$$

where X and Y are variable names indexed with integral indexes over a convex bounded polyhedron $D_p \subset \mathbf{Z}^n$, called the *index domain*. Its bounds are characterized by the *size parameter* $p \in \mathbf{Z}^q$. Any variable Y occuring on the right hand side of such an equation can be either a data variable or a result variable (identical to X or defined by another equation). The computation function f is any O(1) function. Function g is an affine function of the form $g(z) = R \cdot z + r$ from $\mathbf{Z}^n$ to $\mathbf{Z}^{n'}$ ($n' \leq n$), called the *index function*.

The dependency analysis automatically extracts from the SARE the *causal dependencies* between its elementary calculation points. Considering equation (E), we characterize the set of points using a given item $Y[\zeta_0]$, $\zeta_0 \in \mathbf{Z}^{n'}$. This set, called the *utilization set*, [1] is defined by :

$$Util_E(Y, \zeta_0) = \{z \in D_p \mid g(z) = \zeta_0\}.$$

Since the index mapping g is an affine function, the solutions to any $g(z) = \zeta_0$ form an affine subspace of dimension $m \leq n$ of $\mathbf{Z}^n$. Therefore $Util_E(Y, \zeta_0)$ which is the intersection of this affine subspace and the convex bounded polyhedron domain D_p, is a *convex bounded polyhedron*. It can be characterized by a set of basis vectors. These vectors are called the *utilization vectors* and are denoted by $\Phi_{E,Y,i}$ for $i = 1, \ldots, m$.

Notice that $Ker(R)$ (where R defines the index function g, $g(z) = R \cdot z + r$) characterizes the linear space of any utilization set associated with variable Y and that the utilization vectors are basis vectors of $Ker(R)$.

[1] Notice that several equations (E) can have $Y[\zeta_0]$ in their right hand side or that several occurrences of an item of Y can appear in the right hand side of an equation (E). In this case there are as many utilization sets $Util_E(Y, \zeta_0)$ as there are occurrences of Y in the right hand sides of the system.

In a SARE a data does not define any *a priori* dependency between calculations. Thus the causal dependencies imposed by the problem are only related to the result variables. For any result item $Y[\zeta_0]$ the set of points which causally depend on point ζ_0 is $Util_E(Y, \zeta_0)$. The causal dependency related to this utilization set is classically expressed by the notion of *dependency vectors* : $\Psi_{Y,g(z)} = z - g(z)$.

The causal dependency associated with a result variable is therefore characterized by the dependency vectors Ψ and by the utilization vectors Φ. The dependency vectors connect a point $\zeta_0 = g(z)$ computing the result item $Y[\zeta_0]$ to all the points $z \in Util_E(Y, \zeta_0)$, that causally depend on ζ_0. Therefore they have the same origin ζ_0 and their extremities belong to the convex bounded polyhedron $Util_E(Y, \zeta_0)$. Thus they define a *cone* [2] which is characterized by its finite set of *extremal vectors*. These extremal vectors are denoted by Ψ_{Y,ζ_0,ex_i} for $i \in \mathbf{N}$.

All points $\zeta_0 = g(z)$ which are origin of a dependency vector Ψ_{Y,ζ_0} form a set. This set is called the *emission set* [3] and defined by :

$$Emit_E(Y) = \{\zeta_0 \in D \mid \exists z \in D_p \text{ such that } g(z) = \zeta_0\} = g(D_p).$$

Since $Emit_E(Y)$ is the image by the affine function g of a convex bounded polyhedron, it is either a *convex bounded polyhedron* or a *lattice*.

The determination of a parallel program from a problem specification in terms of recurrence equations consists in a space-time mapping. The scheduling function specifies the instant of execution of any elementary computation point z of the problem. An affine schedule is a function from $D \subset \mathbf{Z}^n$ to $\mathbf{N}$ defined by $t(z) = \lambda \cdot z + \alpha$ where $\lambda \in \mathbf{Z}^n$ and $\alpha \in \mathbf{Z}$. Vector λ is orthogonal to hyperplanes of $\mathbf{Z}^n$ such that all the points belonging to one of them have the same execution instant. We call these hyperplanes the *timing surfaces*. Notice that $Ker(\lambda)$ has dimension $n-1$ and is the linear space of any scheduling surface.

Recall that to satisfy the causal dependencies, any affine timing function must be such that its linear part λ satisfies $\lambda \cdot \Psi_{ex} > 0$ for all the extremal vectors Ψ_{ex} associated with the problem. Recall also that the only efficient affine timing functions are constant, i.e. vector λ does not depend on the size parameter.

The allocation function specifies the virtual processor on which any elementary computation point is executed. A linear allocation projects the domain along a unimodular vector called the *allocation direction*. It is denoted by $\xi \in \mathbf{Z}^n$. The schedule and the allocation must satisfy the following classical property.

Property 1 *Let λ be the linear part of an affine timing function and ξ be the allocation direction. These vectors must satisfy $\lambda \cdot \xi \neq 0$.*

Recall that a SARE is characterized by a set of variables, divided into two classes, the data variables and the result variables. Efficient space-time transformations for data variables strongly rely on alignment of data arrays, while efficient transformations for result variables are achieved with alignment of the computations. Alignement of data arrays is based on space transformations characterized by allocation-per-variable functions ([3]). We only consider in this paper compiling for result variables and we show how the characteristics of the target architecture (especially its communication primitives) constrain the determination of both the schedule and the allocation of the computations.

[2] There are as many Ψ-cones as there are result variables Y in the right hand side of any equation (E).

[3] When a result variable Y occurs several times in equation (E), there are as many emission sets as there are occurrences of Y in the right hand side of (E).

3: Compiling a dependency using broadcast primitives

A broadcast primitive consists in one processor sending a value or a message to a set of other processors which receive it simultaneously. It can be realized either on a grid of processors or along only one direction.

Let us consider one dependency associated with result variable $Y[\zeta_0]$. Its utilization set $Util_E(Y, \zeta_0)$ is characterized by its basis of utilization vectors denoted $\Phi_{E,Y,i}$ for $i = 1, \ldots, m$, with $m < n$ where n is the dimension of the problem. Notice that we consider neither the case $m = 0$, nor the case $m = n$. The former case corresponds to a uniform dependency for which there is no utilization vector and for which the notion of broadcast is irrelevant. The later case corresponds to the situation where the whole domain uses the same occurrence of $Y[\zeta_0]$. It usually does not happen in practical cases.

A full broadcast will be implemented for this dependency if all the utilization points associated with $Y[\zeta_0]$ are executed at the same time step, i.e. if they belong to the same timing surface. Hence the space-time transformation must be determined by selecting the schedule first. The allocation can then be chosen arbitrarily under the condition stated by property 1.

The determination of the timing function depends on the dimension of the utilization set. Two cases must be distinguished.

3.1: Utilization set of dimension $n-1$

Since the timing surfaces also have dimension $n-1$, the utilization sets and the timing surfaces must coincide to guarantee a full broadcast. This is stated in the following property.

Property 2 *Let $Y[\zeta_0]$ be a result item characterized by a $(n-1)$-dimensional utilization set $Util_E(Y, \zeta_0)$. The affine timing function ensuring a full broadcast is characterized by a vector λ such that $Ker(\lambda) = Ker(R)$.*

3.2: Utilization set of dimension $m < n-1$

A full broadcast will be obtained if all the utilization points belong to the same timing surface. This gives some freedom to determine the timing function, as described by the next property.

Property 3 *Let $Y[\zeta_0]$ be a result item characterized by a m-dimensional utilization set $Util_E(Y, \zeta_0)$ with $1 \leq m < n-1$. An affine timing function ensuring a full broadcast [4] is such that its linear part λ verifies $\lambda \cdot \Phi_{E,Y,i} = 0$ for $i = 1, \ldots, m$.*

Whatever the dimension of the utilization set is, an affine timing function resulting in a full broadcast of the item $Y[\zeta_0]$ can be determined as long as the causal constraint is not violated. This is expressed by the following property.

Property 4 *Let $Y[\zeta_0]$ be a result item characterized by a m-dimensional utilization set $Util_E(Y, \zeta_0)$ with $1 \leq m \leq n-1$ and m utilization vectors $\Phi_{E,Y,i}, i = 1, \ldots, m$. There exists an affine timing function ensuring a broadcast of $Y[\zeta_0]$ if and only if the extremal vectors Ψ_{ex} associated with $Y[\zeta_0]$ are such that $\Psi_{ex} \neq \sum_{i=1}^{m} \alpha_i \Phi_{E,Y,i}$ for any constants $\alpha_i \in \mathbf{Z}$.*

---

[4] Another way to express this condition is that λ must be such that $Ker(R) \subset Ker(\lambda)$. Such a vector λ always exists since $Ker(R)$ has dimension $m < n-1$.

Proof. Any timing function must satisfy the causal constraints : $\lambda \cdot \Psi_{ex} > 0$ for all the extremal vectors associated with the considered dependency. Let the dependency vectors be such that $\Psi_{ex} = \sum_{i=1}^{m} \alpha_i \Phi_{E,Y,i}$. Any broadcast timing verifies $\lambda \cdot \Phi_{E,Y,i} = 0$ and would result in $\lambda \cdot \Psi_{ex} = 0$. Therefore if $\Psi_{ex} = \sum_{i=1}^{m} \alpha_i \Phi_{E,Y,i}$ then no broadcast timing exists. Let the dependency vectors be such that $\Psi_{ex} \neq \sum_{i=1}^{m} \alpha_i \Phi_{E,Y,i}$. Since the utilization vectors must verify $\lambda \cdot \Phi_{E,Y,i} = 0$, they can always be completed to form a basis of $Ker(\lambda)$ in such a way that $\Psi_{ex} \notin Ker(\lambda)$. Therefore if $\Psi_{ex} \neq \sum_{i=1}^{m} \alpha_i \Phi_{E,Y,i}$ then it is always possible to find a broadcast timing. $\square$

When the target machine only offers 1-dimensional broadcast primitives, the timing function must be such that the intersection between a timing surface and the utilization set (whatever its dimension is) is reduced to one line.

Property 5 *Let $Y[\zeta_0]$ be a result item characterized by a m-dimensional utilization set $Util_E(Y, \zeta_0)$ with $1 \leq m \leq n-1$. An affine timing function ensuring a one-dimensional broadcast is such that its linear part λ verifies $Ker(R) \cap Ker(\lambda)$ has dimension 1.*

In this case this dependency will be implemented by as many 1-dimensional broadcast primitives as there are lines parallel to $Ker(R) \cap Ker(\lambda)$ in the utilization set. To minimize this number and therefore minimize the number of associated communications, one can add other constraints to the one stated in property 5 to determine a schedule. Such constraints could for instance take into account the shape of the utilization set.

4: Compiling a dependency using only local primitives

When the only primitives available on an architecture implement local communications, the number of communications will be minimized if localization is enforced as much as possible. The amount of localization depends on the allocation. Therefore in this context, the space-time transformation must be determined by selecting first the allocation direction and then the affine schedule.

A broadcast is avoided if the intersection of any timing surface with the utilization set $Util_E(Y, \zeta_0)$ is reduced to one point. Since a timing surface has dimension $(n-1)$, this is only possible if the utilization set has dimension 1. For a utilization set of higher dimension, this intersection would be reduced to one point only if the timing function would depend on the parameter size. In this case, the following property ensure the absence of broadcast.

Property 6 *Let Y be a result variable characterized by a 1-dimensional utilization set and one utilization vector $\Phi_{E,Y}$. There will be no broadcast associated with this variable if and only if the linear part λ of the constant schedule is such that $\lambda \cdot \Phi_{E,Y} \neq 0$.*

When the utilization set has dimension larger than one, there does not exist any constant affine schedule that completely removes the broadcast. A broadcast will always exist on smaller-dimensional subsets of the utilization set. If a complete broadcast removal is required, domain transformations are necessary to reduce the dimension of the utilization set by introducing a partial sequentialization of its computations. This problem is related to the study realized in [14] where domain transformations were introduced to deal with problems for which no constant affine timing function exists. It requires a specific study and is out of scope of this paper. Therefore in the following we only consider compiling local communications for 1-dimensional utilization sets.

To ensure localization as much as possible, the allocation direction must project on the same processor as many utilization points as possible. If they are mapped on the same processor, the value $Y[\zeta_0]$ will be used locally for the different computations. This situation occurs when the allocation direction is the basis vector of the utilization set.

Property 7 *Let $Y[\zeta_0]$ be a result item characterized by a 1-dimensional utilization set $Util_E(Y, \zeta_0)$ and one utilization vector $\Phi_{E,Y}$. The allocation direction ξ minimizing the number of communications related to item $Y[\zeta_0]$ is such that $\xi = \pm\Phi_{E,Y}$.*

Notice that since the timing function is such that $\lambda \cdot \Phi_{E,Y} \neq 0$ (by property 6), the condition between the timing and allocation functions stated by property 1 always holds. The number of local communications is deduced from the following property.

Property 8 *Let $Y[\zeta_0]$ be a result item characterized by a 1-dimensional utilization set $Util_E(Y, \zeta_0)$ and one utilization vector $\Phi_{E,Y}$. Let L be the line containing $Util_E(Y, \zeta_0)$. If the allocation direction ξ is such that $\xi = \pm\Phi_{E,Y}$, then the number of communications is*

$$nbcom = \begin{cases} 0 & \textit{if } \zeta_0 \in L \\ 1 & \textit{otherwise} \end{cases}$$

Proof. If $\zeta_0 \in L$, the computation associated with ζ_0 takes place on the same processor as all the computations of $Util_E(Y, \zeta_0)$. Hence there is no induced communication. If $\zeta_0 \notin L$, ζ_0 is not aligned with the points of L, there is one communication required from the processor computing ζ_0 to the processor computing all the utilization points. □

Because of other dependencies associated with a problem, one may finally choose as allocation direction a vector that does not verify property 7 for the considered dependency, i.e. that does not minimize the corresponding number of communications. In this case the number of communications is expressed by the following property.

Property 9 *Let $Y[\zeta_0]$ be a result item characterized by a 1-dimensional utilization set $Util_E(Y, \zeta_0)$ and one utilization vector $\Phi_{E,Y}$. Let Ψ_{ex} be one of its extremal dependency vectors. If the allocation direction ξ is such that $\xi \neq \Phi_{E,Y}$, then the number of communications is*

$$nbcom = \begin{cases} card(Util_E(Y, \zeta_0)) - 1 & \textit{if } \xi = \pm\Psi_{ex} \\ card(Util_E(Y, \zeta_0)) & \textit{otherwise} \end{cases}$$

Proof. Since $\xi \neq \Phi_{E,Y}$, the different utilization points will be allocated to different processors and each of these processors will require the communication of $Y[\zeta_0]$. Therefore the number of communications is $card(Util_E(Y, \zeta_0))$, except if the emission point and one of the utilization points are allocated to the same processor. In this case the number of communications is $card(Util_E(Y, \zeta_0)) - 1$. This situation occurs when the allocation direction ξ is colinear to one of the dependency vectors associated with $Y[\zeta_0]$. In practice, one chooses as allocation direction one of the extremal dependency vectors to guarantee a set of regular and local connections. □

Notice that the $c = card(Util_E(Y, \zeta_0))$ or $card(Util_E(Y, \zeta_0)) - 1$ send primitives do not need to take place on the same processor. Let P_{emit} be the processor that computes $Y[\zeta_0]$. The value it computes must be sent to all the processors computing a point $z \in Util_E(Y, \zeta_0)$. These processors are different since $\xi \neq \pm\Phi_{E,Y}$ and they will use value $Y[\zeta_0]$ at different consecutive instants since $\lambda \cdot \Phi_{E,Y} \neq 0$ (property 6). Therefore the emission processor P_{emit} only needs to send value $Y[\zeta_0]$ to the first processor that will use it. This processor will then send it to the second one that needs it, etc. The c communications can therefore be located on c different processors (using usually neighbor-to-neighbor connections). The distribution of these communications result in a small overhead.

5: Space-time transformation for a SARE

The two previous sections describe how to compile one result dependency associated with a problem in two different contexts (broadcast or local communication primitives). We now consider a problem defined by a SARE. It is usually characterized by several result dependencies. This section discusses how to determine an efficient space-time transformation associated with such a problem in the context of efficient code generation for data parallel programs. The objectives of such a code generation are :

- determine a fast affine schedule, i.e. a schedule that minimizes the global computation time of the algorithm. This is realized using as much as possible the broadcast capabilities (if any) of the target architecture.
- determine a linear allocation that minimizes the number of communications, i.e. localizes as much as possible the data on the processors.

Notice that the best space-time transformation for one dependency may not be (and usually is not) optimal for another dependency. Consider for instance a problem characterized by two result dependencies with 1-dimensional utilization sets. Let Φ_1 and Φ_2 be the respective utilization vectors. In the context of broadcast primitives use, the most efficient schedule for the first dependency is defined by $\lambda \cdot \Phi_1 = 0$ (property 3). The most efficient one relatively to the second dependency should satisfy the same constraint. If the two utilization vectors are not colinear, these two constraints can not hold simultaneously. The best one can hope is to choose a timing function realizing one of the two broadcasts and select as allocation a vector minimizing the number of communications on the other dependency.

The idea is therefore to propose a framework to analyze the different dependencies of a problem in order to determine one or several space-time transformations that are globally efficient. It can be briefly described by the following process.

If broadcast primitives are available

- determine the schedule first by implementing broadcast communications for as many dependencies as possible.
  – consider all the result dependencies with m-dimensional utilization sets ($m > 1$) which satisfy property 4. This property guarantees there exists a schedule with broadcast according to the considered dependency.
  – for each selected dependency, determine one or several timing vectors λ satisfying property 2 or 3 (depending on the dimension of the utilization set).
  – verify that the so-obtained timing vectors are valid for all the dependencies of the problem, i.e. that they respect the causal constraints.
  – keep only the valid timing vectors.
  – if only one-dimensional broadcast primitives are available, then restrict the determination of the schedule according to property 5.
- determine then the allocation by enforcing localization as much as possible.
  – for each efficient schedule determined as mentioned above, consider all the dependencies that do not result in a broadcast.
  – determine one or several allocation vectors ξ that satisfy property 7 (i.e. that localize the data) for as many dependencies as possible without violating property 1 ($\lambda \cdot \xi \neq 0$).

If only local primitives are available

- recall that no affine timing exists if the considered problem is characterized by one or more utilization sets of dimension larger than one.
- determine the allocation first by localizing as many result dependencies as possible. The allocation direction must satisfy property 7 for the largest number of dependencies.
- determine then the schedule. It must be characterized by a constant vector λ verifying the causal constraints, property 6 ($\lambda \cdot \Phi \neq 0$) and property 1 ($\lambda \cdot \xi \neq 0$).

All the pairs (λ, ξ) obtained by such a process can be considered as efficient according to the criteria described in this paper. One then has to generate the code associated to each of these space-time transformations.

The code generation mechanisms are based on the application of the timing and allocation functions to the utilization and dependence vectors of the problem. Due to lack of space this process is not detailed in this paper. We simply present it on two examples. These mechanisms are similar to the one related to systolic array synthesis and have been fully presented in [13]. They also have been fully described [15] for the code generation and data compiling from SURE.

Example 1. Let us consider the problem defined by the following SARE:

$$x(i,j) \;=\; f(x(i,j-1), x(j,j-1)) \quad 2 \leq i \leq n,\ 1 \leq j \leq i-1 \tag{1}$$

where f is any O(1) computation function, and n is an integer constant defining the size parameter of the problem. The initialization is given by $x(i,0) = a_i, 1 \leq i \leq n$ where the values a_i are data. The result is the variable x defined by $x_i = x(i, i-1), 1 \leq i \leq n$.

The dependency analysis gives the following information. There are two occurrences of result variable x in the right hand side of equation (1). The first occurrence is characterized by $Util_1(x,(i_0,j_0)) = \{(i_0, j_0+1)\}$. The utilization set is reduced to one point and this dependency is defined by a constant dependency vector $\Psi_1 = (0,1)$ associated with the emission set $Emit_1(x) = \{(i_0,j_0) \mid 2 \leq i_0 \leq n,\ 1 \leq j_0 < i_0 - 1\}$. The second occurrence of x is characterized by the utilization set $Util_2(x,(i_0,i_0-1)) = \{(i,i_0) \mid i_0+1 \leq i \leq n\}$. This set has dimension 1 and it is characterized by one utilization vector $\Phi_x = (1,0)$. The corresponding dependency vectors are defined by $\Psi_{2,(i_0,i_0-1)} = (i-i_0, 1)$ with $i_0+1 \leq i \leq n$. The corresponding emission set is $Emit_2(x) = \{(i_0, i_0-1) \mid 2 \leq i_0 \leq n-1\}$.

In the context of broadcast primitives, one must first consider the second dependency. From property 2, the affine timing function ensuring broadcast is characterized by $\lambda = (0,1)$. This schedule guarantees one broadcast communication per time step on the whole architecture. The first dependency yields no communication if the allocation direction is $\xi = \Psi_1 = (0,1)$ (property 9 extended to 0-dimensional utilization sets). Any other allocation direction will result in one communication per time step on each active processor for the second dependency. Let us choose $\xi = (1,1)$ as allocation direction. This direction is interesting because it projects all the emission points of $Emit_2(x)$ on the same processor, called P_{Emit} in the following. Therefore the broadcast primitive occurs only in the code of this processor.

The resulting architecture contains $n-1$ processors numbered $P_1, \ldots, P_{n-1}$ with $P_1 = P_{Emit}$. Each processor realizes the same operation at each step, except at its first activation where its computation uses initial values. When applying the synthesis method presented in [13] and [15] one gets the following code for these processors where reg is an internal register :

```
Processor P_1(P_Emit)
Begin
    x = f(a_2, a_1)
    reg = x
    send x to P_2 ... P_{n-1}
    loop
        receive x_1
        x = f(x_1, reg)
        reg = x
        send x to P_2 ... P_{n-1}
    end
end
```

```
Processor P_i(i = 2 ... n-1)
Begin
    x = f(a_{i+1}, a_1)
    send x to P_{i-1}
    loop
        receive x_1
        receive x_2
        x = f(x_1, x_2)
        send x to P_{i-1}
    end
end
```

In the context of local primitives only, one deduces from property 7 that the optimal allocation direction relatively to the second dependency is $\xi = \Phi_x = (1, 0)$. Since the point $(i_0, i_0 - 1)$ is not aligned with the set $Util_2(x, (i_0, i_0 - 1)) = \{(i, i_0) \mid i_0 + 1 \leq i \leq n\}$ there is one communication per time step on the whole architecture relatively to this dependency (property 8). This allocation direction induces one communication per time step on each active processor relatively to the first dependency of x. The timing function must respect the causal constraints $(\lambda \cdot \Psi > 0)$ and must induce no broadcast communications ($\lambda \cdot \Phi_x \neq 0$, property 6). In the following we consider $\lambda = (1, 1)$ as timing function.

The allocation direction yield $n - 1$ processors numbered $P_1 \ldots P_{n-1}$. Except the first processor which computes its function using data values (denoted a_i), the other processors are identical. Their first activation corresponds to the computation of the emission points for which they each receive a value from their neighbor. This value (called x_2 in the program below) is then used locally for all their other computations. The code of the processors is :

```
Processor P_1
Begin
    x = f(a_2, a_1)
    send x to P_2
    for i = 3 to n
        x = f(a_i, a_1)
        send x to P_2
    end
end
```

```
Processor P_i(i = 2 ... n-1)
Begin
    receive x_2
    receive x_1
    x = f(x_1, x_2)
    send x to P_{i+1}
    loop
        receive x_1
        x = f(x_1, x_2)
        send x to P_{i+1}
    end
end
```

6: The LL^t Cholesky factorization

Let us consider the classical problem of the Cholesky factorization and show how to determine efficient space-time transformations as well as the corresponding generated code.

Let A be an $n \times n$ symmetric positive definite matrix. The Cholesky factorization calculates a lower triangular matrix L such that $A = LL^t$. It is defined by the following SARE.

$$\begin{aligned} L(i,j,k) &= L(i,j,k-1) - L(i,k+1,k) \times L(j,k+1,k) \\ &\qquad 2 \leq i \leq n, \quad 2 \leq j \leq i, \quad 0 \leq k \leq j-2 \end{aligned} \tag{2}$$

$$
\begin{aligned}
L(i,j,k) &= L(i,j,k-1)/L(j,j,j-1)) && (3)\\
&\qquad 2 \le i \le n, \quad 1 \le j \le i-1, \quad k=j-1 &&\\
L(i,j,k) &= L(i,j,k-1)^{1/2} && (4)\\
&\qquad 1 \le i \le n, \quad j=i, \quad k=i-1 &&
\end{aligned}
$$

The initialization is given by $L(i,j,-1) = A_{ji}$ $1 \le i \le n, 1 \le j \le i$. The resulting matrix L is composed of the elements $L_{ij} = L(i,j,j-1)$ $1 \le i \le n, 1 \le j \le i$. The problem is characterized by 3 convex subdomains D_1, D_2 and D_3 respectively associated with equations (2), (3) and (4).

$$
\begin{aligned}
D_1 &= \{(i,j,k) \in \mathbf{Z}^3 \mid 2 \le i \le n, \quad 2 \le j \le i, \quad 0 \le k \le j-2\},\\
D_2 &= \{(i,j,j-1) \in \mathbf{Z}^3 \mid 2 \le i \le n, \quad 1 \le j \le i-1\},\\
D_3 &= \{(i,i,i-1) \in \mathbf{Z}^3 \mid 1 \le i \le n\}.
\end{aligned}
$$

In the 3 subdomains there is a constant dependence vector $\Psi_0 = (0,0,1)$ due to the use of $L(i,j,k-1)$ in all three equations. In equation (1) element $L(i,k+1,k)$ defines emission set $Emit_1(L)$ and utilization set $Util_1(L,\zeta_0)$, while element $L(j,k+1,k)$ defines emission set $Emit_2(L)$ and utilization set $Util_2(L,\zeta_0)$. In equation (2) we call $Emit_3(L)$ and $Util_3(L,\zeta_0)$ the emission and utilization sets due to element $L(j,j,j-1)$. Let $\zeta_0 = (i_0, j_0, k_0)$. These different sets are:

$$
\begin{aligned}
Emit_1(L) &= \{(i_0,j_0,j_0-1) \mid 2 \le i_0 \le n, 1 \le j_0 \le i_0-1\} = D_2\\
Util_1(L,\zeta_0) &= \{(i_0,j,j_0-1) \mid j_0+1 \le j \le i_0\}\\
Emit_2(L) &= \{(i_0,j_0,j_0-1) \mid 2 \le i_0 \le n, 1 \le j_0 \le i_0-1\} = D_2\\
Util_2(L,\zeta_0) &= \{(i,i_0,j_0-1) \mid i_0 \le i \le n\}\\
Emit_3(L) &= \{(i_0,i_0,i_0-1) \mid 1 \le i_0 \le n-1\} = D_3 - \{(n,n,n-1)\}\\
Util_3(L,\zeta_0) &= \{(i,i_0,i_0-1) \mid i_0+1 \le i \le n\}
\end{aligned}
$$

The corresponding utilization and dependency vectors are respectively defined by :

$$
\begin{array}{ll}
\Phi_1 = (0,1,0) & \Psi_1 = (0, j-j_0, 0) \quad with \quad 1 \le j-j_0 \le i_0 - j_0\\
\Phi_2 = (1,0,0) & \Psi_2 = (i-i_0, i_0-j_0, 0) \quad with \quad 0 \le i-i_0 \le n-i_0\\
\Phi_3 = (1,0,0) & \Psi_3 = (i-i_0, 0, 0) \quad with \quad 1 \le i-i_0 \le n-i_0
\end{array}
$$

In the context of broadcast primitives, one first try to determine a schedule $\lambda = (\lambda_1, \lambda_2, \lambda_3)$. Since $\Psi_1 = \alpha\Phi_1$ and $\Psi_3 = \alpha\Phi_3$, property 4 is not satisfied for the two corresponding dependencies. Therefore no affine timing function ensuring broadcast for these two dependencies exists. Property 4 is verified for the second dependency characterized by Ψ_2 and Φ_2 and one deduces from property 3 that a timing ensuring broadcast for this dependency must verify $\lambda \cdot \Phi_2 = 0$, i.e. $\lambda_1 = 0$. Unfortunately any such vector λ does not respect the causal constraint on the third dependency : $\lambda \cdot \Psi_3 = 0$. Therefore it is not possible to schedule this problem with a function ensuring some broadcast.

We therefore now concentrate on the determination of the allocation in order to minimize the number of communications. The schedule will be determined later according to the causal constraints and to property 6. Among the three one-dimensional utilization sets, two are characterized by utilization vectors $\Phi = (1,0,0)$. Therefore the allocation direction ensuring localization as much as possible is $\xi = (1,0,0)$ (property 7). The affine timing function must verify the causal constraints and property 6. This yields the conditions $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 > 0$. The optimal timing function is therefore $\lambda = (1,1,1)$.

For this space-time transformation characterized by the timing function $t(i,j,k) = i + j + k$ and the allocation function $alloc(i,j,k) = (j,k)$, the resulting solution is an array of processors $P_{x,y}, 1 \leq x \leq n, 0 \leq y \leq x-1$. The code is obtained from the applications of these two functions to the dependency and utilization vectors of the problem. Notice that the processors $P_{x,0}$ receive initial values.

```
Processor P_{x,y}
(1 ≤ x ≤ n, y = x − 1)
% executes first equation (4)
% and then equation (3)
Begin
    receive ℓ_1
    ℓ = ℓ_1^{1/2}
    reg = ℓ
    loop
        receive ℓ_1
        ℓ = ℓ_1/reg
        send ℓ to P_{x+1,y}
    end
end
```

```
Processor P_{x,y}
(2 ≤ x ≤ n, 0 ≤ y ≤ x − 2)
% executes equation (2)
Begin
    receive ℓ_1
    receive ℓ_2
    ℓ = ℓ_1 − ℓ_2 × ℓ_2
    reg = ℓ_2
    send ℓ to P_{x,y+1}
    send ℓ_2 to P_{x+1,y}
    loop
        receive ℓ_1
        receive ℓ_2
        ℓ = ℓ_1 − ℓ_2 × reg
        send ℓ to P_{x,y+1}
        send ℓ_2 to P_{x+1,y}
    end
end
```

7: Conclusion

This paper deals with two important issues in data-parallelism : the efficient use of communication primitives (in particular the broadcast ones) and the minimization of the number of communications by localizing the data on the processors in the context of result variables compiling. These two issues concur to generate fast and efficient data-parallel algorithms.

We have shown in this paper that they influence the determination of both the schedule and the allocation and that the determination of efficient space-time transformations requires compromises between optimizing the schedule and optimizing the allocation.

This paper shows how to optimize these two criteria when dealing with one result dependency and presents empirical solutions to the general problem of finding efficient space-time transformations for a problem with several result dependencies. It shows the need of a systematic study that will help to understand better the relationships between the schedule and the allocation.

This work should be extended to integrate already well-known techniques of data arrays alignment in order to tackle the general problem of efficient compiling for both data and result variables. It should also be extended to compiling techniques constrained by a fixed number of processors.

References

[1] Banerjee U., Eigenman R., Nicolau A., Padua A., *Automatic program parallelization.* Proceedings of the IEEE, **81(2)**, pp. 211-243, 1993.

[2] Cappello P.R., *A spacetime-minimal systolic array for matrix product.* In Systolic Array Processors, J. McCanny et al. Editors, Prentice Hall, pp.347–356, 1989.

[3] Darte A., Robert Y., *A graph-theoretic approach to the alignment problem.* Research report 93-20, LIP-ENS Lyon, 1993.

[4] D'Hollander E., *Partitioning and labeling of loops by unimodular transformations.* IEEE Transactions on parallel and distributed systems. **3(4)**, pp. 465–476, 1992.

[5] Dowling M., *Optimal code parallelization using unimodular transformations.* Parallel computing, **16**, pp. 157-171, 1990.

[6] Fortes J.A.B., Fu K.S., Wah B.W., *Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays.* Int. Conf. on Acoustics, Speech and Signal Processing,1987.

[7] Hillis W.D., Steele G.L., *Data parallel algorithms.* Communications of the ACM, **29(12)**, pp. 1170–1183, 1986.

[8] Gupta M., Banerjee P., *A methodology for high-level synthesis of communication for multicomputers.* 6th ACM Int. Conf. on Supercomputing, 1992.

[9] Karp R.M., Miller R.E., Winograd S., *The Organization of Computations for Uniform Recurrence Equations.* JACM, **14(3)**, 1967.

[10] Kuck D.J., Kuhn R.H., Leasure B.R., Wolfe M.J., *The structure of an advanced retargetable vectorizer.* Supercomputers : design and applications tutorial, pp. 967-974, 1984.

[11] Kung H.T., Leiserson C.E., *Systolic arrays for VLSI.* Sparse matrix proceedings, SIAM, pp. 245-282, 1978.

[12] Lamport L., *The parallel execution of DO loops.* Communications of the ACM, **17(2)**, pp. 83–93, 1974.

[13] Mongenet C., Clauss Ph., Perrin G.R., *Geometrical tools to map systems of affine recurrence equations on regular arrays.* Acta Informatica, **31**, pp. 137–160, 1994.

[14] Mongenet C., *Affine timings for systems of affine recurrence equations.* Conference on Parallel Architectures and Languages Europe, PARLE'91, LNCS 505, Springer verlag, pp. 236–251, 1991.

[15] Mongenet C., *Data compiling for systems of uniform recurrence equations.* Parallel processing letters, To appear.

[16] Quinton P., Van Dongen V., *The Mapping of Linear Recurrence Equations on Regular Arrays.* The Journal of VLSI Signal Processing, **1**, pp 95–13, 1989.

[17] Rajopadhye S., *Synthesizing systolic arrays with control signals from recurrence equations.* Distributed Computing, **3**, pp. 88–105, 1989.

[18] Ramanujam J., *Non-unimodular transformations of nested loops.* Proc. of Supercomputing 92, pp. 214–223, 1992.

[19] Ramanujam J., Sadayappan P., *Compile-time techniques for data distribution in distributed memory machines.* IEEE Transactions on parallel and distributed systems, **2(4)**, pp. 472–482.

[20] Wolf M., Lam M., *A loop transformation theory and an algorithm to maximize parallelism.* IEEE Transactions on parallel and distributed systems, **2(4)**, pp. 452-471, 1991.

[21] Zima H.P., Bast H.J., Gerndt H.M., *SUPERB ; a tool for semi-automatic MIMD SIMD parallelization.* Parallel Computing, **6**, pp. 1-18, 1988.