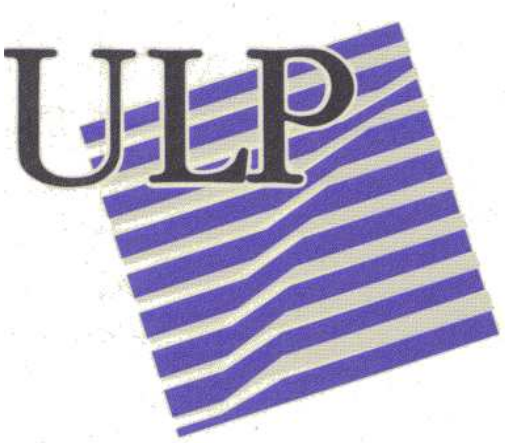

ICPS

Informatique et
Calcul Parallèle
de Strasbourg

Publication 94-08

Optimal Mapping of Systolic Algorithms by Regular Instruction Shifts

Ph. Clauss and G.-R. Perrin

Published in ASAP'94, August 22-24, 1994, San Francisco (USA), p. 224-235, IEEE Computer Society Press

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

Optimal Mapping of Systolic Algorithms by Regular Instruction Shifts

Philippe Clauss
University Robert Schuman
F-67082 Strasbourg
clauss@caseb.u-strasbg.fr

Guy-René Perrin
University of Franche-Comté
F-25030 Besançon
perrin@caseb.u-strasbg.fr

Abstract

This paper addresses the problem of determining efficient mappings of systems of affine recurrence equations into regular arrays, in a nearly space-optimal fashion. A new non linear allocation technique is presented : the *Instruction Shift*. It allows to synthesize planar regular arrays without increasing the initial linear schedule. This technique is illustrated with the LL^t Cholesky factorization.

Keywords

Affine recurrences, Linear or non linear allocations, Space-optimal arrays, Systolic arrays.

1: Introduction

Distributed memory multiprocessors are attractive to their scalability, flexibility and performance but suffer from a lack of efficient methods for mapping algorithms. A large number of algorithms in signal and data processing applications can be written in the form of nested loops [13] and more generally in the form of systems of affine recurrence equations [11]. These formalisms and the problems they involve were first adressed by Karp et al. [7] and Lamport [9]. Later on, many works [15, 12, 8, 10] on the design of systolic arrays, proposed a linear transformation of the iteration space into a $space \times time$ domain.

One of the major drawbacks of these methods is the low processing element (PE) utilization occurring in the solutions. These methods suffer of the way allocation techniques are defined as linear mappings. Some further works focused on this problem in order to increase the solution efficiency, by defining non linear allocations. Non linear allocations may not suit VLSI arrays, which have elementary PEs computing an identical and fine grain instruction, whereas they can convene for todays parallel distributed memory computers. These facilities justify the use of non linear allocation techniques.

The allocation technique proposed in this paper allows to map systems of affine recurrence equations into regular processor arrays, in a space-optimal or nearly space-optimal fashion. These solutions have regular topologies in the sense that any PE communicates

only with its neighbours. This property ensures that these solutions can easily be mapped into a mesh connected array, without loss of interest about their efficiency. Moreover, the derived solutions can be linearly time-optimal since our mapping technique does not affect the initial linear schedule.

This technique, called the *Instruction Shift technique*, is a sort of reallocation of an initial virtual systolic solution, but it differs from classical compression techniques, such as clustering [3, 1, 5], since it applies even on arrays whose PEs are active at any instant between their first and last calculations. This technique also differs from recent works on recurrences or nested loops mapping : in these studies indeed, a good utilization of PEs is obtained in some cases but these methods lead generally to empirical results. For example, Peir and Cytron in [11], or D'Hollander in [6] propose a partitioning into independent groups of variable instances and an allocation of each group into a processor space of the same dimension. Unfortunately many recurrences (such as Cholesky factorization, LU decomposition, Gauss elimination, etc.), yield only one partition.

2: Systems of Affine Recurrence Equations

Recurrence equations can be considered as abstract forms of nested loops, in which the only dependencies are flow-dependencies, since such statements apply the single assignment rule. In practice, we assume that dependencies are affine functions of the iteration vector and the iteration space Dom is a convex polytope of Z^n, where n is the depth of the nest.

Definition 1 *An affine recurrence equation is of the form* $X(\vec{z}) = f(..., Y(g(\vec{z})), ...)$, *where* $\vec{z} \in Dom(X) \subset Z^n$, g *is an affine function from* Z^n *to* Z^m *and* f *has a constant computation time.* [1]

Vertices of the dependence graph associated with equations are the integral points inside Dom, and dependencies can be defined by dependence vectors $\vec{d} = \vec{z} - g(\vec{z})$ in Z^n. A space-time allocation consists in substituting some $S \times T$ space for Dom, where S is $(n-1)$-dimensional and T is 1-dimensional. The computation of S lies on a projection direction $\vec{\xi}$: the allocation is then a linear function $alloc(\vec{z}) = \sigma\,\vec{z}$, which maps the iteration space onto S. The allocation $(n-1, n)$-matrix σ is determined from a pivoting algorithm which computes an orthogonal basis containing $\vec{\xi}$. The schedule is an affine function $t(\vec{z}) = \vec{\lambda} \cdot \vec{z} + \alpha$ on the iteration space. In this paper we only consider practical cases where the schedule does not depend on the variable X. Vector $\vec{\lambda}$ is orthogonal to hyperplanes of Z^n, the *timing surfaces*, whose points have the same execution instant. So, the space $\times$ time transformation is expressed as matrix $\begin{pmatrix} \vec{\lambda} \\ \sigma \end{pmatrix}$. Such a transformation is valid if the schedule satisfies $\vec{\lambda} \cdot \vec{d} > 0$ for any dependence vector $\vec{d}$, and if the allocation is such that $\vec{\lambda} \cdot \vec{\xi} > 0$, since all the points of Dom aligned on $\vec{\xi}$ are executed into the same PE and, therefore, cannot be executed at the same instant.

Example 1 The Cholesky factorization.
Let A *be a* $N \times N$ *symmetrical positive definite matrix. The Cholesky factorization yields* $A = LL^t$, *where* L *is lower triangular :*

$$L(i,j,k) \;=\; L(i,j,k-1) - L(i,k+1,k) \times L(j,k+1,k)$$

[1] In the following, Dom is the union of the $Dom(X)$ for all variables X defined in the nest. In practice, we assume that variables are fully indexed on Z^n, i.e. $m = n$ and that Dom is a convex polytope depending on some size parameters.

$$
\begin{aligned}
& & & 1 \leq i \leq N, \quad 1 \leq j \leq i, \quad 0 \leq k \leq j-2 \\
L(i,j,k) &= L(i,j,k-1)/L(j,j,j-1) & & \\
& & & 1 \leq i \leq N, \quad 1 \leq j \leq i-1, \quad k = j-1 \\
L(i,j,k) &= L(i,j,k-1)^{1/2} & & 1 \leq i \leq N, \quad j = i, \quad k = i-1
\end{aligned}
$$

The input equation is $L(i,j,-1) = A_{ji}$, $1 \leq i \leq N, 1 \leq j \leq i$, *and the iteration space is* $\{(i,j,k) \in Z^3 \mid 1 \leq i \leq N,\ 1 \leq j \leq i,\ 0 \leq k \leq j-1\}$. *Regarding dependencies a systolic solution can be synthesized. It is characterized by schedule* $\vec{\lambda} = (1,1,1)$ *and allocation direction* $\vec{\xi} = (1,0,0)$ *defining for example allocation matrix* $\sigma = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. ♢

Such arrays present many qualities of regularity and locality of the connections. However, by studying PEs' activity through the geometrical modeling described in the next section, it is shown that the disadvantage of most systolic mappings is that they do not have an optimal parallelism rate. Moreover, the program computed by each PE may have a too much fine grain for a mapping onto a distributed memory computer. The performances and the memory storage capabilities of such array processors allow more sophisticated programs. That is the reason why, in order to keep their qualities but overcome their disadvantages we aim to map these solutions on more general distributed memory networks with the objective of minimizing the virtual processor count and keeping the regularity.

3: Geometrical modeling

The aim of this section is to recall some notions about a geometrical modeling of PEs' activity. The reader can refer to [2, 3] for further investigations. The following notions come from well-known mathematical results [14] :

- Assuming the iteration space *Dom* is a convex polytope of Z^n defined by a non redundant set of integral inequations $\{\vec{z} \in Z^n \mid A\vec{z} \leq \vec{b}\}$, where A is totally unimodular, any line A_i of matrix A defines the normal vector to a facet, which is contained in the hyperplane which saturates the i^{th} constraint, i.e. defined by the equation $A_i \cdot \vec{z} - b_i = 0$,
- By projecting *Dom* along an allocation direction, the activity of any resulting PE is defined by the points of an *allocation segment* $Dom_s = \{\vec{z} \in Dom \mid \sigma\,\vec{z} = s\}$, whose extremities characterize facets of *Dom* : they correspond either to the first or to the last computation on this PE. However it may happen that these facets belong to several parallel integer hyperplanes and are *thick facets*. To simplify the presentation, we consider from now on that there are no such thick facets.

Definition 2 *Assuming the schedule and the allocation are such that* $\vec{\lambda} \cdot \vec{\xi} > 0$, *we call* first facets *of the iteration space Dom, facets whose points* $\vec{z}$ *are such that* $\vec{z} - \vec{\xi} \notin Dom$. *We call* last facets *of the iteration space Dom, facets whose points* $\vec{z}$ *are such that* $\vec{z} + \vec{\xi} \notin Dom$.

The intersections between a timing surface and the first (respectively last) facets characterize all the PEs having their first (resp. last) activation at the same instant. These intersections are either faces of dimension $(n-2)$, or the facets themselves. In the following we consider the former case.

We denote as F_i (resp. L_j) a first (resp. last) facet defined by equation $A_i \cdot \vec{z} - b_i = 0$ (resp. $A_j \cdot \vec{z} - b_j = 0$), and call it the i^{th} first facet (resp. the j^{th} last facet). The intersection of this facet with the timing surface at instant t_0 is denoted as F_{i,t_0} (resp. L_{j,t_0}). Their

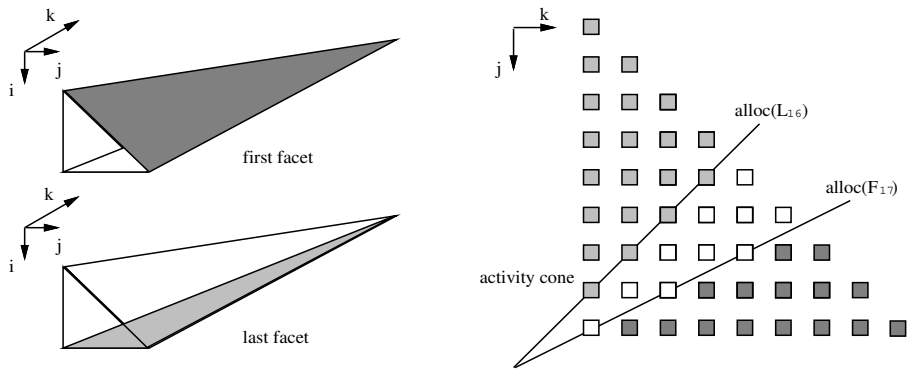

Figure 1. The Cholesky factorization

union for all first and last facets are respectively denoted as F_{t_0} and L_{t_0}. Projections of these sets F_{t_0} and L_{t_0} along the allocation direction $\vec{\xi}$, on the initial architecture, define the activity period of every PE. More precisely we introduce the concept of activity cone.

Definition 3 *Hyperplanes $alloc(F_{t_0+\vec{\lambda}.\vec{\xi}})$ and $alloc(L_{t_0})$ define the so called* activity cones *of the array. They delimit three areas of PEs at any instant t_0 : the dead PEs, the possibly active ones and the previously unused ones.*

This geometrical characterization of active PEs is going to be used to define a specific allocation technique whose objective is to map the systolic solution into regular arrays with minimization of the processor count.

Definition 4 *For a given schedule, we call* parallelism rate *the maximum number of simultaneous calculations $pp = max_{\,1 \leq t_0 \leq \tau}\ p(t_0)$, where $p(t_0)$ is the number of simultaneous calculations at instant t_0 and τ is the latency of the algorithm.*

Value $p(t_0)$ can be determined from the first and last activity instant of every PE [2, 3].

Example 2 The Cholesky factorization (continued).
The allocation direction $\vec{\xi} = (1,0,0)$ defines one first facet F and one last facet L (in this order for the considered schedule $\vec{\lambda} = (1,1,1)$). Inequations defining Dom characterize their normal vectors $(-1,1,0)$ and $(1,0,0)$. Sets F_{t_0} and L_{t_0} are lines generated by vectors $(1,1,-2)$ and $(0,1,-1)$. The resulting activity cone is then defined by its extremal vectors $(1,-2)$ and $(1,-1)$ and shifts along vector $(0,1)$ at each time step. The parallelism rate, depending on N, can then be determined : for $N = 9$ we have $pp = 12$. $\diamondsuit$

4: The instruction shift technique

The problem of minimizing the processor count from systolic arrays was already studied in several recent works. The clustering technique [1, 2, 3, 5] applies only on systolic arrays whose PEs are not active at any instant between their first and last calculations but ensures regular solutions which can be space-optimal in some cases [2]. The serializing technique [2, 3, 4] applies in any case, but space-optimal solutions can only be determined from initial PEs as fast as the global clock. It leads to nonplanar cylindrical, orbital or more generally toroidal solutions.

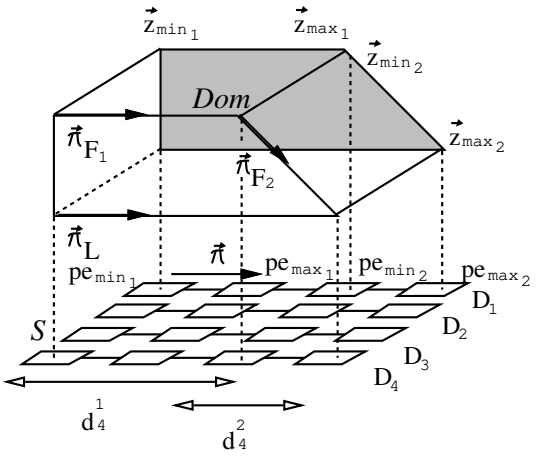

Figure 2. Mathematical objects used by the instruction shift technique

All these methods compress an initial solution in a static way. The technique we present is a new approach in which calculations are reallocated. It applies in any case and results in planar and regular solutions. Moreover, these solutions are a nice compromise between regularity and space-optimality as shown in the following.

4.1: Principles of the method

Systolic arrays are not space-optimal, i.e. the surface of their activity cones is much smaller than the surface of the array. The largest surface defined by the cones is sufficient indeed to compute the initial systolic algorithm. The *instruction shift* technique consists in considering these cones as an array, i.e. in mapping the calculations of the initial array into the architecture defined by the cones.

The systolic array is first partitioned into parallel lines of PEs generated by some vector $\vec{\pi}$ called the *partition direction*. Then, when a PE appears inside the activity cone, its calculation is allocated to the first resulting PE in the direction $\vec{\pi}$. Its next calculations are allocated to the same resulting PE until an other initial PE appears inside the activity cone. Its following calculations are allocated to the next resulting PE in the partition direction, and so on. So, this technique is founded on shifts of successive calculations of any initial PE into a line of resulting PEs. Notice that this technique ensures that the neighbouring of initial PEs is nearly the same in the resulting array, and then ensures local connections. It consists in the following steps :

- Partition the systolic array in a set of parallel lines of PEs along the direction $\vec{\pi}$,
- For each line D, compute the maximum number $act(D)$ of simultaneously active PEs,
- For each line D, shift and allocate calculations into $act(D)$ PEs.

Notice that PEs of any line D result from the projection of a 2-dimensional hyperplane which intersects with the first and last facets of the iteration space Dom. These intersections are line segments generated by n-dimensional unimodular vectors denoted respectively by $\vec{\pi}_{F_i}$ and $\vec{\pi}_{L_j}$. The application of the allocation function *alloc* on these vectors is the partition direction, i.e. $\vec{\pi} = alloc(\vec{\pi}_{F_i}) = alloc(\vec{\pi}_{L_j})$.

4.2: Determination of the partition direction

The first step consists in determining a convenient partition direction $\vec{\pi}$. In order to compute the maximum number $act(D)$ of simultaneously active PEs in each line D, we use the following proposition which links vectors $\vec{\pi}_{F_i}$, $\vec{\pi}_{L_j}$ and $\vec{\pi}$.

Proposition 1 *Vectors $\vec{\pi}_{F_i}$, $\vec{\pi}_{L_j}$ and $\vec{\pi}$ are such that*

$$\vec{\pi}_{F_i} = \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{\pi} \\ 0 \end{pmatrix} \qquad \vec{\pi}_{L_j} = \begin{pmatrix} \sigma \\ A_j \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{\pi} \\ 0 \end{pmatrix}$$

for any i^{th}-first and any j^{th}-last facets.

Proof – The partition direction is such that $\vec{\pi} = \sigma\,\vec{\pi}_{F_i}$. Assuming the i^{th} line A_i of A defines the normal vector to the i^{th} first facet F_i, we have $A_i \cdot \vec{\pi}_{F_i} = 0$. This leads to $\begin{pmatrix} \sigma \\ A_i \end{pmatrix} \cdot \vec{\pi}_{F_i} = \begin{pmatrix} \vec{\pi} \\ 0 \end{pmatrix}$. Since matrix $\begin{pmatrix} \sigma \\ A_i \end{pmatrix}$ is non singular, the result comes. ♢

Definition 5 *We call sub-line d^i the segment of PEs in a line D resulting from the projection of points of the first facet F_i. In the same way, sub-line d^j of D results from the projection of points of the last facet L_j.*

Proposition 2 *Let us consider the f first facets of Dom. The number $nf(t_0)$ of PEs on some line D which have their first activation before some instant t_0 (included) is :*

$$\begin{array}{ll} \forall t_0 \in [1 \ldots \vec{\lambda} \cdot \vec{z}_{max_0}[& nf(t_0) = 0 \\ \forall t_0 \in [\vec{\lambda} \cdot \vec{z}_{min_i} \ldots \vec{\lambda} \cdot \vec{z}_{max_i}[& nf(t_0) = (t_0 - \vec{\lambda} \cdot \vec{z}_{min_i}) \div \vec{\lambda} \cdot \vec{\pi}_{F_i} \\ & \qquad + nf(\vec{\lambda} \cdot \vec{z}_{max_{i-1}} - 1) + 1 \\ \forall t_0 \in [\vec{\lambda} \cdot \vec{z}_{max_f} \ldots \tau[& nf(t_0) = nf(\vec{\lambda} \cdot \vec{z}_{max_f} - 1) + 1 \end{array}$$

where $\forall i \in [1 \ldots f]$, z_{min_i} and z_{max_i} are points in the first facet F_i allocated into the two extremal PEs pe_{min_i} and pe_{max_i} of sub-line d^i, i.e.

$$\vec{z}_{min_i} = \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} pe_{min_i} \\ b_i \end{pmatrix} \qquad \vec{z}_{max_i} = \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} pe_{max_i} \\ b_i \end{pmatrix}$$

Proof – Vector $\vec{\pi}_{F_i}$ characterizes line segments in the first facet F_i. For each line D its projection into the processor space results in a sub-line d^i. Therefore, the number of PEs in d^i which have their first activation before t_0 is equal to the number of points of the line segment occurring in the time interval $[1 \ldots t_0]$. Let $\vec{z}_{min_i}$ and $\vec{z}_{max_i}$ be the intersection points between the line segment and two edges of F_i and such that $t(\vec{z}_{min_i}) \leq t(\vec{z}_{max_i})$. Assuming that $t_0 \geq \vec{\lambda} \cdot \vec{z}_{min_i}$, the point count in $[1 \ldots t_0]$ reduces to the point count in $[\vec{\lambda} \cdot \vec{z}_{min_i} \ldots t_0]$. Assuming $t_0 < \vec{\lambda} \cdot \vec{z}_{max_i}$, this count results in $(t_0 - \vec{\lambda} \cdot \vec{z}_{min_i}) \div \vec{\lambda} \cdot \vec{\pi}_{F_i} + 1$. The result comes for the whole line D by considering all sub-lines d^j, $j < i$, for $t_0 = \vec{\lambda} \cdot \vec{z}_{max_j} - 1$. ♢

The number $nl(t_0)$ of PEs on a line D which have their last activation before some instant t_0 (included) is determined in the same way. Then, this result comes :

Proposition 3 *The maximum number of simultaneously active PEs in a line D is*

$$act(D) = MAX_{1 \leq t_0 \leq \tau} \; (nf(t_0) - nl(t_0 - 1))$$

Example 3 The Cholesky factorization (continued).
The previous systolic mapping is partitioned following the vector $\vec{\pi} = (1,1)$. *Vectors* $\vec{\pi}_F$ *and* $\vec{\pi}_L$ *are determined using Proposition 1 :*

$$\vec{\pi}_F = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad \vec{\pi}_L = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

One can then compute $act(D_q)$ *for any line* D_q *of the partition :*

q	1	2	3	4	5	6	7	8	9
$act(D_q)$	3	3	3	2	2	2	1	1	1

♢

Notice that the number of PEs resulting from such a mapping is equal to $\sum_q act(D_q)$, where q is a $(n-2)$-dimensional index depending on the partition direction. Therefore, this direction can be chosen in order to minimize this value. This yields the following non-linear integer programming problem :

$$min\ \{\textstyle\sum_q act(D_q) \mid MAX_{1 \leq t_0 \leq \tau}\ (nf_q(t_0) - nl_q(t_0 - 1))\}$$

where functions nf_q and nl_q are the functions nf and nl associated with each line D_q.

4.3: Synthesis aspects of the instruction shift technique

4.3.1: The allocation function *Shift*

Any solution designed by the instruction shift technique is represented in a $(n-1)$-dimensional space. So, the coordinates of any resulting PE can be expressed in an orthogonal basis $(\vec{\pi}, \vec{\pi}_2, \ldots, \vec{\pi}_{n-1})$ including vector $\vec{\pi}$. Let pe be the coordinates of any PE of the initial systolic array. This PE belongs to a line D_q, where index q is the $(n-2)$-uple $(\vec{\pi}_2 \cdot pe, \ldots, \vec{\pi}_{n-1} \cdot pe)$. Moreover, let us precise that pe computes a calculation which occurs in the first facet F_i, by subscripting its coordinates with a little i. Mapping a calculation of such a pe_i depends on its position inside its associated activity cone when it computes the considered calculation, as this proposition says.

Proposition 4 *The position of any initial PE* pe_i *inside its associated cone, computing a calculation at instant* t_0 *is defined by*

$$pos(pe_i, t_0) = (t_0 - f(pe_i)) \div \vec{\lambda} \cdot \vec{\pi}_{F_i} + 1$$

where $f(pe_i) = t\left(\begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} pe_i \\ b_i \end{pmatrix}\right)$ *is the first activity instant of this initial PE.*

Proof – The partition direction $\vec{\pi}$ is such that $\vec{\pi} = alloc(\vec{\pi}_{F_i})$. Therefore, the first activations of two successive PEs of the same line are time-spaced by $\vec{\lambda} \cdot \vec{\pi}_{F_i}$ instants. So, the position of any initial PE inside its associated cone changes rythmically every $\vec{\lambda} \cdot \vec{\pi}_{F_i}$ instants. ♢

Proposition 5 *The allocation function* $Shift$ *determining the PEs' integral coordinates in the resulting processor space is :*

$$Shift(pe_i, t_0) = (pos(pe_i, t_0), q)$$

Notice that since $pe_i = alloc(\vec{z})$ and $t_0 = t(\vec{z})$, this allocation function can easily be expressed depending directly on the integral coordinates of any iteration vector $\vec{z}$ in Dom. This expression defines a non linear allocation of the iteration space, denoted as $Shift_alloc$ in the following. This allocation does not increase the complexity of the control flow, since the generated code can contain the rythmically repeated calculations of every resulting PE.

Example 4 The Cholesky factorization (continued).
The solution is represented in a 2-dimensional space. The coordinates of any resulting PE are defined in the basis $\vec{\pi} = (1,1), \vec{\pi}_2 = (1,-1)$. *Let* (x,y) *the coordinates of any initial PE. Its first activity instant is*

$$f(x,y) = t\left(\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix}^{-1} \cdot \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}\right) = 2x + y - 1$$

Since $\vec{\lambda}\cdot\vec{\pi}_F = 3$ *the position inside the cone of this PE computing at instant* t_0 *is* $pos(x,y,t_0) = ((t_0 - 2x - y + 1) \div 3) + 1$. *The allocation function* $Shift$ *is then :*

$$Shift(x,y,t_0) = (((t_0 - 2x - y + 1) \div 3) + 1, x - y)$$

This function can be expressed as a non linear allocation depending on the integral coordinates (i,j,k) *of any point of the iteration space :*

$$Shift_alloc(i,j,k) = (((i-j) \div 3) + 1, j - k)$$

$\diamondsuit$

Now the architecture is designed. The next step consists in transforming the connections between systolic PEs into connections between PEs in the resulting processor space.

4.3.2: Connection synthesis

Definition 6 *Let* pe_{out} *be an emitting PE and* t_{out} *an emission instant, and let* pe_{in} *be an associated receiving PE and* t_{in} *its reception instant. A connection between those PEs is the couple* $(\vec{c}, t_{\vec{c}})$, *with* $\vec{c} = pe_{in} - pe_{out}$ *and* $t_{\vec{c}} = t_{in} - t_{out}$.

The application of the allocation function $Shift$ to any initial connection $(\vec{c}_{init}, t_{\vec{c}_{init}})$ results in connections $(\vec{c}_{res}, t_{\vec{c}_{res}})$ in the processor space. Since shifts do not affect the initial schedule : $t_{\vec{c}_{res}} = t_{\vec{c}_{init}} = t_{in} - t_{out}$. Moreover, $\vec{c}_{res} = Shift(pe_{in}, t_{in}) - Shift(pe_{out}, t_{out})$.

Proposition 6 *Resulting connections* $\vec{c}_{res} = (\vec{c}_{res_1}, \vec{\pi}_2 \cdot \vec{c}_{init}, \ldots, \vec{\pi}_{n-1} \cdot \vec{c}_{init})$ *are such that*

$$\left\lceil \frac{t_{\vec{c}_{init}} - \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{c}_{init} \\ 0 \end{pmatrix} + 1}{\vec{\lambda}\cdot\vec{\pi}_{F_i}} \right\rceil - 1 \leq \vec{c}_{res_1} \leq \left\lfloor \frac{t_{\vec{c}_{init}} - \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{c}_{init} \\ 0 \end{pmatrix} - 1}{\vec{\lambda}\cdot\vec{\pi}_{F_i}} \right\rfloor + 1$$

Proof –

$$\begin{aligned} \vec{c}_{res} &= Shift(pe_{in}, t_{in}) - Shift(pe_{out}, t_{out}) \\ &= Shift(pe_{out} + \vec{c}_{init}, t_{out} + t_{\vec{c}_{init}}) - Shift(pe_{out}, t_{out}) \\ &= (\alpha \div \vec{\lambda}\cdot\vec{\pi}_{F_i} + 1, \vec{\pi}_2 \cdot (pe_{out} + \vec{c}_{init}), \ldots, \vec{\pi}_{n-1} \cdot (pe_{out} + \vec{c}_{init})) \\ &\qquad -((t_{out} - f(pe_{out})) \div \vec{\lambda}\cdot\vec{\pi}_{F_i} + 1, \vec{\pi}_2 \cdot pe_{out}, \ldots, \vec{\pi}_{n-1} \cdot pe_{out}) \end{aligned}$$

where

$$\begin{aligned}\alpha &= t_{out} + t_{\vec{c}_{init}} - f(pe_{out} + \vec{c}_{init}) \\ &= t_{out} + t_{\vec{c}_{init}} - f(pe_{out}) - \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{c}_{init} \\ 0 \end{pmatrix}\end{aligned}$$

since for any $(n-1)$-dimensional vector $\vec{v}$, $f(pe_i + \vec{v}) = f(pe_i) + \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ A_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{v} \\ 0 \end{pmatrix}$.
Let $\beta = \vec{\lambda} \cdot \vec{\pi}_{F_i}$ and $\gamma = t_{out} - f(pe_{out})$. Therefore, $\vec{c}_{res_1}$ can be written as $\left\lfloor \frac{\alpha}{\beta} \right\rfloor - \left\lfloor \frac{\gamma}{\beta} \right\rfloor$. So $\frac{\alpha-\gamma+1}{\beta} - 1 \leq \vec{c}_{res_1} \leq \frac{\alpha-\gamma-1}{\beta} + 1$. Since $\vec{c}_{res_1}$ is integer, it is bounded as follows :

$$\left\lceil \frac{\alpha - \gamma + 1}{\beta} \right\rceil - 1 \leq \vec{c}_{res_1} \leq \left\lfloor \frac{\alpha - \gamma - 1}{\beta} \right\rfloor + 1$$

$\diamondsuit$

Example 5 The Cholesky factorization (continued).
Here are the initial connections in the systolic array and the ones resulting of the shift instruction technique :

connection	$\vec{c}_{init}$	$t_{\vec{c}_{init}}$	$\vec{c}_{res}$	$t_{\vec{c}_{res}}$
1	$(0,0)$	*1*	$(0,0)$ $(1,0)$	*1*
2	$(1,0)$	*1*	$(-1,1)$ $(0,1)$	*1*
3	$(0,1)$	*1*	$(0,-1)$	*1*

$\diamondsuit$

The third synthesis step now consists in distributing the input data and defining connections to bring them from the array border to the convenient PEs.

4.3.3: Input data-flow synthesis

The instruction shift technique distributes the calculations of any initial systolic PE on the partition of the processor space, even for input operations. A first naive solution could consist in considering that any PE of the resulting array can operate a data input. This solution is convenient in a data-parallel programming point of view, by defining a dedicated data alignment. However, it is undesirable for VLSI arrays which require input flows on the borders. So, input data flows are reorganized to meet this condition, by expansion along the partition direction $\vec{\pi}$.

According to Proposition 4 calculations of an initial PE are allocated to different resulting PEs, by clusters of size $\vec{\lambda} \cdot \vec{\pi}_{F_i}$. Considering any initial input PE, a series of $\vec{\lambda} \cdot \vec{\pi}_{F_i}$ inputs must be received by any resulting PE in its line. The data flow direction and the time-space between clusters of inputs are synthesized from the following observations : let us consider two clusters of θ inputs to be received by two neighbouring PEs in the same line, i.e. two PEs spaced by $\vec{\pi}$. The first active one receives its first input at instant δ_1, its next one at instant $\delta_1 + 1$, etc., and its last one at instant $\delta_1 + \theta - 1$. The second active PE must receive its first input at instant $\delta_2 = \delta_1 + \theta$. So, the resulting input flow organization depends on its direction : if it is equal to $\vec{\pi}$, the last data of the first cluster pushes the first data of the second one. This case is then wrong. Conversely, if it is opposite to $\vec{\pi}$, the two input clusters are time-spaced by a one top delay : this case is then correct.

Every initial line D_q results, by instruction shift, in $act(D_q)$ PEs. This number depends

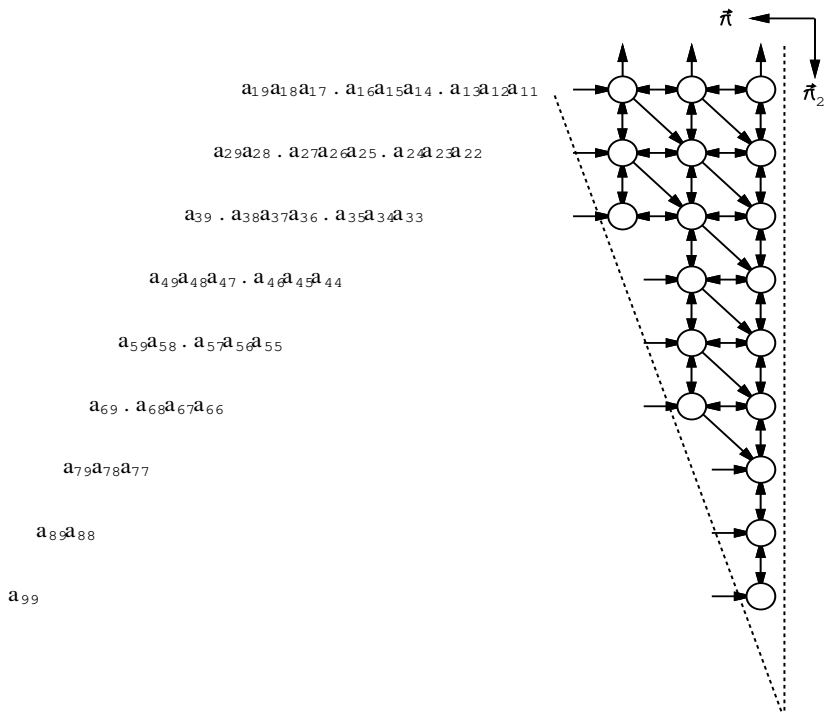

Figure 3. Resulting mapping by shifting the Cholesky factorization

on the considered line, so the initial input PEs result in a non constant number of PEs. In order to ensure correct synchronization, delays have to be added on some input flows. For an input flow associated with a line D_q, this delay is equal to $act - act(D_q)$, where act is the maximum of the values $act(D_q)$ on the partition.

Example 6 The Cholesky factorization (continued).
The input flows are divided into clusters of size 3. *Connections of direction opposite to* $\vec{\pi}$, *occurring between the resulting PEs, are added to the previous ones. Since* $act = 3$, *the input schedule is modified by adding* 1 *top on flows associated with lines* D_4, D_5 *and* D_6, *and* 2 *tops on flows associated with lines* D_7, D_8 *and* D_9. *Moreover, on each flow, clusters are spaced by a* 1 *more top. The resulting mapping is shown on Fig.3.* ♢

4.4: Optimality analysis

First remind that instruction shift maintains all schedule initial properties : if the systolic array is time-optimal, so is the processor array. An optimality analysis of arrays obtained by instruction shift will then focus only on space-optimality.

By definition of the instruction shift technique, any initial systolic PE pe_i is allocated into a number of resulting PEs equal to $\left\lceil \frac{d(pe_i)}{\vec{\lambda}\cdot\vec{\pi}_{F_i}} \right\rceil$, where $d(pe_i)$ is the number of calculations, i.e. the activity duration, of pe_i. Therefore, considering all the pe_is of any line D_q, the calculations they achieve need at least a number of resulting PEs equal to $np_q = MAX_i \left\lceil \frac{d(pe_i)}{\vec{\lambda}\cdot\vec{\pi}_{F_i}} \right\rceil$.

From this observation, one can deduce that if $act(D_q) < np_q$ the instruction shift technique results in a number of PEs per line greater than the minimum required. Anyway,

the number $np = \sum_q np_q$ of resulting PEs of such a solution can be compared with the one obtained by a classical clustering :

- The worst case occurs when the number of initial PEs in line D_q is less than np_q : the technique increases the number of PEs and its application becomes inopportune.
- Favourable cases occur when the partition direction $\vec{\pi}$ defines lines such that $act(D_q) \geq np_q$. In these cases, every line D_q of the initial systolic array is transformed into a line of PEs whose count is optimal and equal to $act(D_q)$. Therefore, each resulting line can be said space-optimal.

The ratio $\Delta = np/pp$, which links the processor count and the optimality of the solution, can be seen as a distance between optimality per line and global optimality. As shown in the following, examples emphasize that Δ is bounded by a constant value whatever the size of the problem parameters. This means that the best partition directions define nice compromises between regular and optimal solutions.

Example 7 The Cholesky factorization (continued).
The following table gives the different values of np_q and $act(D_q)$ for the example which is a favourable case :

q	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
np_q	*3*	*3*	*3*	*2*	*2*	*2*	*1*	*1*	*1*
$act(D_q)$	*3*	*3*	*3*	*2*	*2*	*2*	*1*	*1*	*1*

More generally values of np_q and $act(D_q)$ depending on q, are $np_q = act(D_q) = \left\lfloor \frac{N-q}{3} \right\rfloor + 1$. The processor count np can then be determined :

$$np = \begin{cases} \frac{N\,(N+3)}{6} & if\ N\ mod\ 3 = 0 \\ \frac{(N+1)\,(N+2)}{6} & otherwise \end{cases}$$

The systolic solution results in $N(N+1)/2$ PEs and the parallelism rate is :

$$pp = \begin{cases} \frac{N\,(N+2)}{8} & if\ N\ mod\ 4 = 0\ or\ 2 \\ \frac{(N+1)^2+4}{8} & if\ N\ mod\ 4 = 1 \\ \frac{(N+1)^2}{8} & if\ N\ mod\ 4 = 3 \end{cases}$$

The ratio $\Delta = np/pp$ can be determined in any case, depending on N. One can then show that $\lim_{N\to\infty} \Delta = 4/3$. The same ratio for the initial systolic array shows that its processor count is 4 times larger than the parallelisme rate, i.e. 3 times larger than the processor count determined by applying the instruction shift technique. ♢

5: Conclusion

The technique presented in this paper is a non linear allocation for mapping systems of affine recurrence equations or nested loops. It is a first proposal of this new technique. Some further works have to be developped in order to synthesize systematically practical solutions. Other partitions of the initial systolic mapping may also be considered : for example, a partition in series of line segments, or in a set of parallel lines, etc.

Moreover, an *active* shift version of this technique, as in the clustering approach, may also be relevant in order to map an algorithm into a fixed number of processors. Such a method could apply by imposing a number of calculations to be allocated into every PE,

larger than the minimum required by the initial linear schedule.

Such an allocation technique can be used as a program design method for massively parallel machines. Since instruction shift results in a high utilization of the target architecture, it addresses the reality of programming distributed memory computers.

References

[1] J. Bu, E.F. Depreterre, and P. Dewilde. A design methodology for fixed-size systolic arrays. *IEEE Int. Conf. on Application Specific Array Processors*, pages 591–602, 1990.

[2] P. Clauss. *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* PhD thesis, Univ. of Franche-Comté, May 1990.

[3] P. Clauss, C. Mongenet, and G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. *IEEE Int. Conf. on Application Specific Array Processors*, pages 591–602, 1990.

[4] P. Clauss, C. Mongenet, and G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the algebraic path problem : A new contribution. *Parallel Computing*, 18:185–194, 1992.

[5] A. Darte. Regular partitioning for synthesizing fixed-size systolic arrays. *Integration, the VLSI Journal*, 12:293–304, 1991.

[6] E. D'Hollander. Patitioning and labelling of index sets in DO loops with constant dependence vectors. In *Int. Conf. on Parallel Processing, 2*, pages 139–144, 1989.

[7] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of ACM*, 14(3):563–590, Juillet 1967.

[8] S.Y. Kung. *VLSI array processors.* Prentice-Hall, 1988.

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

[10] C. Mongenet. *Une méthode de conception d'algorithmes systoliques : résultats théoriques et réalisation.* PhD thesis, INPL, May 1985.

[11] J.K. Peir and R. Cytron. Minimum distance: a method for partitioning recurrences for multiprocessors. *IEEE Transactions on Computers*, 38(8):1203–1211, aug 1989.

[12] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrence equations. *IEEE, Int. Symp. on Computer Architecture*, pages 208–214, 1984.

[13] S.K. Rao and Th. Kailath. Regular iterative algorithms and their implementation on processor arrays. *Proceedings of IEEE*, 76(4):259–269, 1988.

[14] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1986.

[15] W. Shang and J.A.B. Fortes. Time optimal linear schedules for algorithms with uniform dependences. In K. Bromley et al. eds., editor, *International Conference on Systolic Arrays*, pages 393–402, 1988.