5

Systolic Petri Nets

Alexandre Abellard and Patrick Abellard
HandiBio EA4322, IUT, Toulon University
France

1. Introduction

In many research fields and applications requiring real time, such as signal, speech or image processing, problems are often characterized by the amount of data to deal with.
However, it can happen that real time constraints are difficult to satisfy without taking into account the intrinsic parallelism of processings to perform. Thus, diverse architectures appeared (SIMD, MIMD...), leading to different classes of architecture. Literature showed the advantages and drawbacks of each of them. Moreover, expensive costs limited their applications during a long time.
Systolic architectures defined by H.T. Kung are a particular class of parallel architectures. They constitute specialized systems characterized by a repetitive structure of identical elementary processors locally and regularly interconnected. Synchronous data circulate through the architecture, which interact at each encounter. Several important difficulties like the central memory sharing, buses access conflicts... can be therefore avoided. All problems are however not systolizable. These networks are designed for the repetitive identical processing of a huge amount of data, which is the case, for example, in many signal and image processing algorithms.
The conception of these networks has been the subject of many studies, the main ones are developped in the first part of this chapter. The limited number of systolic processors available on the market was problematic for the development of methodologies for a long time. Now, programmable components enable to be free from this problem. To ease their implementation, we developped a methodology based on a formal and universal tool (Petri Nets) that is developped in the second part of the paper.

2. Definitions

2.1 Systolic architecture

This architecture is considered as an intermediate between data flow and pipeline. It has been introduced in (Kung, 1982) and is made of a set of processors or processing cells locally interconnected. Each cell can run a simple or complex operation and links between cells are established so as to minimize the associated paths complexity.

Just like in a pipeline structure, information move in a cascade scheduling form. Communications between outside environment and systolic network are established with peripheral cells that constitute the network I/O ports.

2.2 Systolic networks properties

Systolic networks have interesting properties("Quinton & Robert, 1991") :
- Data flow coming from the environment are intensively used
- Networks association is made easier thanks to structure cascadability
- Elementary cells are not complex
- Data flow are simple and regular
- A set of elementary processings is performed synchronously on different cells

These main properties enable to simplify their implementation on VLSI once an automated and rigorous conception method has been defined.
Several kinds of nets exist, according to different basic cells (Fig. 1) : the linear array, orthogonal array and hexagonal array. All can be unidirectional or bidirectional. In this paper, we will mostly focus on linear arrays since it better suits our application, Fig. 1 therefore does not show all possible propagation directions in arrays.

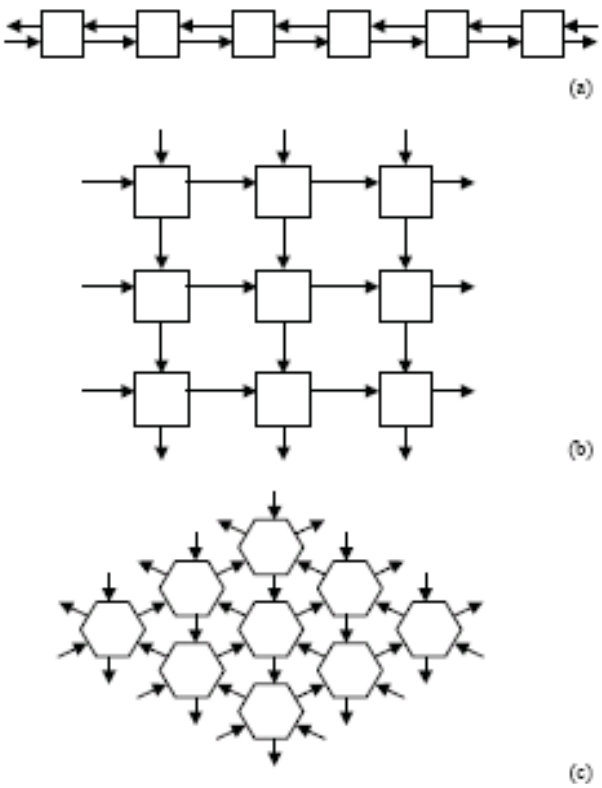


Fig. 1. Basic systolic architectures : (a) linear array, (b) squared array, (c) hexagonal array

These architectures take advantage of the massive parallelism encountered in processing applications (Johnson & Hurson, 1993). They only need a minimum of operators (Sousa, 1998) and memory accesses (Kung, 1988) thanks to a very efficient communications system (Lim & Swartzlander, 1996a). Their combination allows to obtain arrays (Lim & Swartzlander, 1996a) that can be used in a wide range of applications : Discrete Fourier Transform (Lim & Swartzlander, 1999b) (Jackson et al., 2004) (Nash, 2005), convolution (Lee & Song, 2003), filtering (Lee & Song, 2004), matrix operations (Yang et al., 2005), dynamic programming (Lee & Song, 2002).... The regularity of their structures facilitate their hardware, implementation, for instance in FPGAs (Mihu et al., 2001) (Nash, 2002) (Castro-Pareja et al., 2004).

2.3 Principles

2.3.1 Example of a linear network

Linear equation solving is done thanks to the following equation:

$$y_i^{k+1} = a_{i,k+1}.x_{k+1} + y_i^k \, , \, 0 \le k \le n\text{-}1, \, 1 \le i \le n \tag{1}$$

Systolic network defined by Kung is established with a group of interconnected processors, each having 3 registers : R_y for y^k, R_a for $a_{i,k}$ and R_x for x. Each register has a connection as input and another one as output. Kung defined 2 kinds of cells : squared (Fig. 2a) and hexagonal (Fig. 2b).

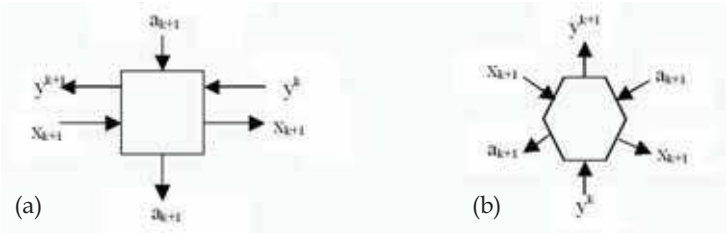


Fig. 2. Square (a) and hexagonal (b) cells

These two kinds of cells work in this operating cycle.
1 – the cell loads inputs y^k, x_{k+1} and $a_{i,k+1}$ in respective registers R_y, R_x and R_a.
2 – y^{k+1} is processed using equation (1)
3 – y_i^{k+1}, x_{k+1} and $a_{i,k+1}$ are transferred to the output
Example of the Matrix-Vector Product (MVP)

$$Y = A \, . \, X \tag{2}$$

Relations to implement are then :

$$y_i^{k+1} = a_{i,\,k+1} \, . \, x_{i,\,k+1} + y_i^k \tag{3}$$
$$y_i^0 = 0 \; ; \; y_i = y_i^W$$

with W=dim(X), 0≤k ≤n-1, 1≤i≤n

For example, with W = 3 :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Equation (3) then gives :

$$y_1 = y_1^0 + y_1^1 + y_1^2 + y_1^3 \Rightarrow y_1 = a_{11}.x_1 + a_{12}.x_2 + a_{13}.x_3$$
$$y_2 = y_2^0 + y_2^1 + y_2^2 + y_2^3 \Rightarrow y_2 = a_{21}.x_1 + a_{22}.x_2 + a_{23}.x_3$$
$$y_3 = y_3^0 + y_3^1 + y_3^2 + y_3^3 \Rightarrow y_3 = a_{31}.x_1 + a_{32}.x_2 + a_{33}.x_3$$

Given an elementary recurrence relation :

$$y_i^{k+1} = a_{i,\,k+1} \,.\, x_{i,\,k+1} + y_i^k$$

successive recurrence steps are performed at consecutive instants. The use of parallelism done via several cells enable to perform a step of many different elementary recurrences at the same time. Systolic network is therefore made of a set of (2.*n*-1) linearly interconnected squared cells, each one receiving y_i^k, x_{k+1} and $a_{i,k+1}$ at each step of time t_j (Fig. 3).

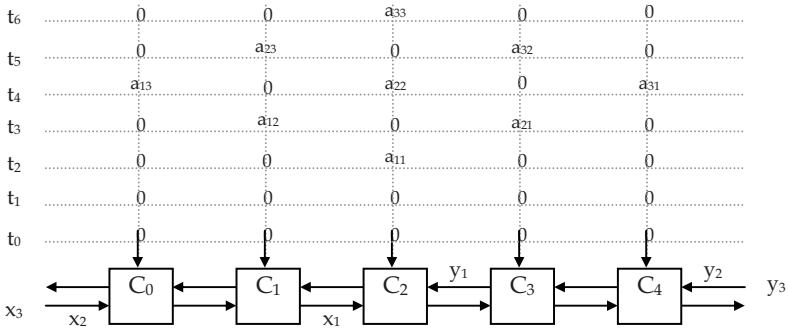


Fig. 3. Linear systolic network of matrix-vector product *Y=A.X*, *n*=3

x_1 is used by C_0 cell at t_0, then is transmitted to C_1 that processes it at t_1 and so on from left to right. A similar processing is done for y_i data from right to left. Fig. 4 shows the detail of data propagation on cells performing $y_i^{k+1} = a_{i,\,k+1} \,.\, x_{i,\,k+1} + y_i^k$.

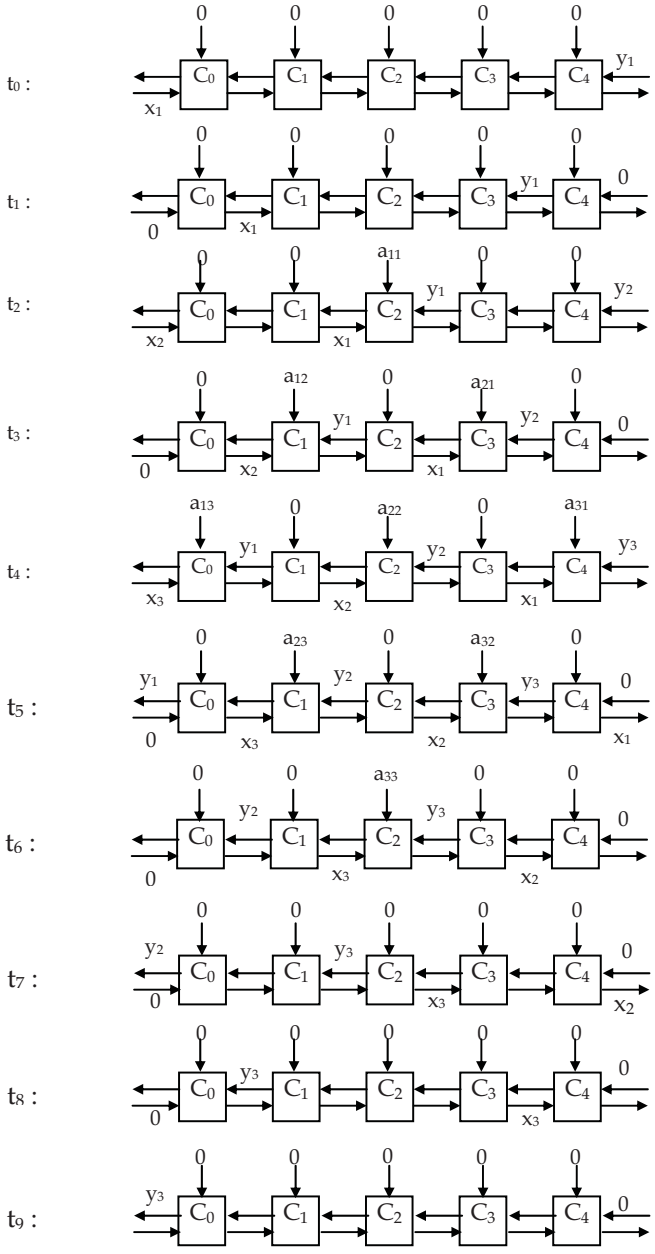


Fig. 4. Linear systolic network processing for matrix-vector product $Y=A.X$ (size 3)

2.3.2 Example of a bi-dimensional network

Consider now the matrix product : C = A.B, being sizes of A (m,n), B (n,p) and C (m,p). Each coefficient of C is processed using :

$$c_{i,j} = \mathrm{Sum}(a_{i,k}.b_{k,j})_{\ k=1..n}\ ,\ 1 \le i \le n,\ 1 \le j \le n,\ 1 \le k \le n \qquad (4)$$

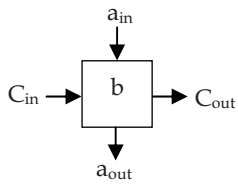


Fig. 5. Elementary cell

$$a_{out} = a_{in}\ ;\ C_{out} = C_{in} + a_{in}.b$$

This relation can be expressed via a recurrence relation like in 2.3.1. Coefficients a_{ik} propagate on j axis, and b_{kj} propagate on i axis. k is a recurrence axis that can be assimilated to a temporal axis. Elementary processings given by (4) are all identical. Network is thus made of a sole kind of cell (Fig. 5) that can be associated in square.

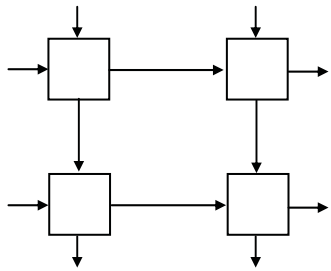


Fig. 6. Example of a 2×2 squared network

Data propagation is detailed on Fig. 7. Other organisation and data propagation possibilities exist. Hexagonal cells can also be used for this processing. In these cells, data propagate in three directions (Fig. 8) so as to be used by neighbouring ones (Fig. 9).

Conceiving systolic networks depends on the problem to be solved and the forced constraints (minimizing number of cells, data flow…). As a consequence, there is no unique method on conception. Several methods exist, some using mathematical equations of the problem to solve, others using problem algorithms. This next section will deal with these questions.

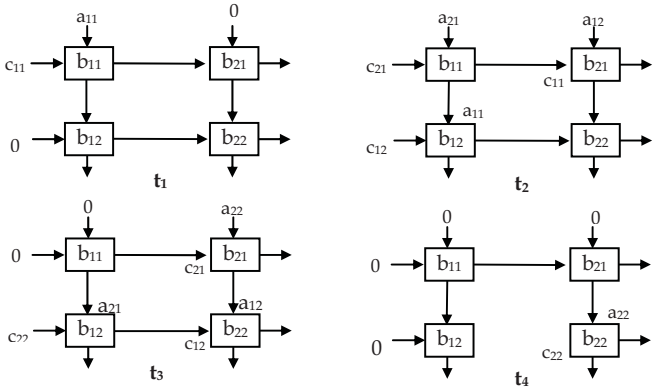
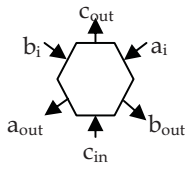


Fig. 7. Data propagation in a squared network



$a_{out} = a_{in}$
$b_{out} = b_{in}$
$c_{out} = c_{in} + a_{in}.b_{in}$

Fig. 8. Elementary hexagonal cell

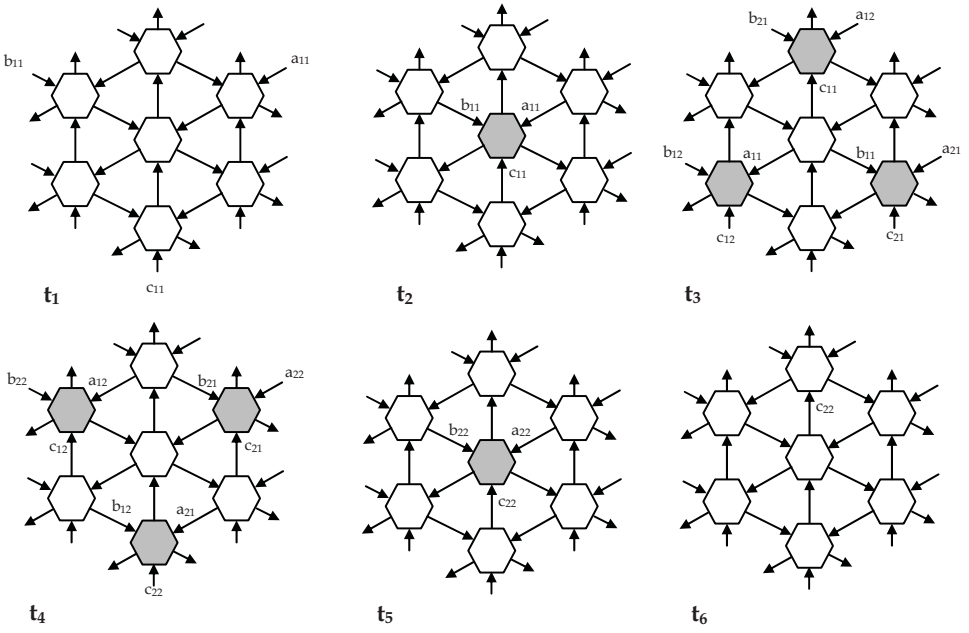


Fig. 9. Hexagonal systolic network operating cycle

3. Equation-solving based methods

Among the various approaches done, the three main ones respectively use recurrent equations, sequential algorithms transformation and fluency graphs.

3.1 Recurrent equations based method

3.1.1 Quinton method

It is based on the use of geometrical domain projection representing the processing to be done so as to define systolic structures (Quinton, 1983). It has three steps :
- Expressing the problem by a set of uniform recurrent equations on a domain $D \subset \mathbf{Z}^n$
- From this set of equations, defining a temporal function so as to schedule processings
- Defining one or several systolic architectures by applying processing allocation functions to elementary cells

These functions are determined by the different processing domain projections.

3.1.1.1 Step 1 : Creating recurrent equations

Be $\mathbf{R}^n$, the n-dimension real numbers space, $\mathbf{Z}^n$ its subset with integer coordinates and $D \subset \mathbf{Z}^n$ the processing domain. On each point z from D, a set of equations $E(z)$ is processed :

$$\begin{aligned} u_1(z) &= f(u_1(z+\theta_1), u_2(z+\theta_2), ..., u_m(z+\theta_m)) \\ u_2(z) &= u_2(z+\theta_2) \\ &... \\ u_m(z) &= u_m(z+\theta_m) \end{aligned} \quad (5)$$

in which vectors $\theta_i \subset \Theta$ called dependency vectors are independent from z. They define which are the values where a point of the domain must take its input values. This system is uniform since θ_I does not depend on z and the couple (D, Θ) represents a dependency graph. Thus, the processing of A and B (2 n×n-matrices) is defined by :

$$c_{ij} = \mathrm{Sum}(a_{ik}.b_{kj})_{k=1..n}\ ,\ 1 \le i \le n,\ 1 \le j \le n$$

It can be defined by the following uniform recurrent equations system :

$$\begin{aligned} c(i,j,k) &= a(i,j,k-1)+a(i,j-1k).b(i-1,j,k) \\ a(i,j,k) &= a(i,j-1,k) \\ b(i,j,k) &= a(i-1,jk) \end{aligned} \quad (6)$$

Several possibilities to propagate data on i, j and k axis exist. a_{ik}, b_{kj} and c_{ij} are respectively independent from j, i and k, the propagation of these 3 parameters can be done following the (i,j,k) trihedron. The processing domain is the cube defined by D = {(i,j,k), 0≤i≤n, 0≤j≤n, 0≤k≤n}. Dependency vectors are θ_a = (0, 1, 0) , θ_b = (1, 0, 0) , θ_c = (0, 0, 1). With n=3, dependency graph can be represented by the cube on Fig. 10. Each node corresponds to a processing cell. Links between nodes represent dependency vectors. Other possibilities for data propagation exist.

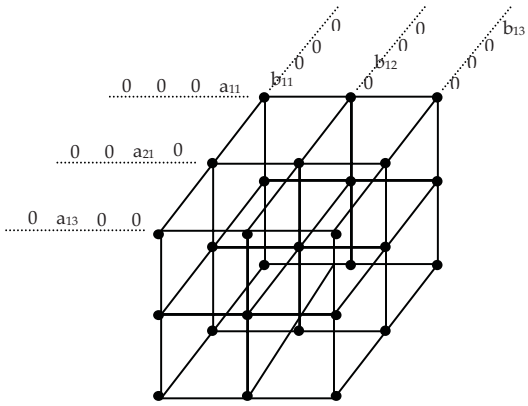


Fig. 10. Dependency domain for matrix product

3.1.1.2 Step 2 : Determining temporal equations

The second step consists in determining all possible time functions for a system of uniform recurrent equations. A time function t is from $D \subset \mathbf{Z}^n \rightarrow \mathbf{Z}^n$ that gives the processing to perform at every moment. It must verify the following condition :

If $x \in D$ depends on $y \in D$, i.e. if a vector dependency $\Theta_i = \overrightarrow{yx}$ exists, then $t(x)>t(y)$.

When D is convex, analysis enables to determine all possible quasi-affine time functions. In this aim, following definitions are used :

- D is the subset of points with integer coordinates of a convex poyedral $\underline{D}$ from $\mathbf{R}^n$.
- $\text{Sum}(\mu_i.x_i)_{i=1...m}$ is a positive combination of points $(x_1, \ldots, x_n)$ from $\mathbf{R}^n$ if $\forall i , \mu_i > 0$
- $\text{Sum}(\alpha_i.x_i)_{i=1...m}$ is a convex combination of $(x_1, \ldots, x_n)$ if $\text{Sum}(\alpha_i)_{i=1...m} = 1$
- s is a summit of $\underline{D}$ if s can not be expressed as a convex combination of 2 different points of $\underline{D}$
- r is a radius of $\underline{D}$ if $\forall x \in \underline{D}$, $\forall \mu_i \in \mathbf{R}^+$, $x+\mu_i.r \in \underline{D}$
- a radius r of $\underline{D}$ is extremal if it can not be expressed as a positive convex combination of other radii of D.
- l is a line of $\underline{D}$ if $\forall x \in \underline{D}$, $\forall \mu_i \in \mathbf{R}$, $x+\mu_i.l \in \underline{D}$
- if $\underline{D}$ contains a line, $\underline{D}$ is called a cylinder

If we limit to convex polyedral domains that are not cylinders, then the set S of summits of $\underline{D}$ is unique as well as the set R of $\underline{D}$ extremal radii. $\underline{D}$ can then be defined as the subset of points x from $\mathbf{R}^n$ with $x = y + z$, y being a convex combination of summits of S and z a positive combination of radii of R.

Definition 1. $T = (\lambda, \alpha)$ is a quasi-affine time function for (D, Θ) if $\forall \theta \in \Theta$, $\lambda^T.\theta \geq 1$, $\forall r \in R$, $\lambda^T.r \geq 0$, $\forall s \in S$, $\lambda^T.s \geq \alpha$

Thus, for the uniform recurrent equations system defining the matrix product, (λ,α) time functions meets the following characteristics :

$$\lambda^T=(\lambda_1, \lambda_2, \lambda_3) \text{ with } \lambda_1 \geq 1, \lambda_2 \geq 1, \lambda_3 \geq 1 \text{ and } \lambda_1 + \lambda_2 + \lambda_3 > 1.$$

A possible time function can therefore be defined by $\lambda^T = (1,1,1)$, with the following 3 radii (1,0,0), (0,1,0) and (0,0,1).

3.1.1.3 Step 3 : Creating systolic architecture

Last step of the method consists in applying an allocation function ξ of the network cells. This function $\xi=a(x)$ from D to a finite subset of $\mathbf{Z}^m$ where m is the dimension of the resulting systolic network, must verify the following condition (t : time function seen on 3.1.1.2) that guarantees that two processings performed on a same cell are not simultaneous :

$$\forall x \in D, \forall y \in D, a(x)=a(y) \Rightarrow t(x) \neq t(y).$$

Each cell has an input port $I(\mu_i)$ and an output port $O(\mu_i)$, associated to each μ_i , defined in the system of uniform recurrent equations. $I(\mu_i)$ of cell C_i is connected to $O(\mu_i)$ of cell $C_{i+a.\theta i}$ and $O(\mu_i)$ of cell C_i is connected to $I(\mu_i)$ of cell $C_{i-a.\theta i}$. Communication time between 2 associated ports is $t(\theta_i)$ time units. For the matrix product previously considered, several allocation functions can be defined. :

- ξ = (0,0,1) or (0,1,0) or (1,0,0), respectively corresponding to a(i,j,k)=k, a(i,j,k)=j, a(i,j,k)=i. Projection of processing domain in parallel of one of the axis leads to a squared shape
- ξ = (0,1,1) or (1,0,1) or (1,1,0), respectively corresponding to a(i,j,k)=j-k, a(i,j,k)=i-k, a(i,j,k)=i-j. Projection of processing domain in parallel of the bisector lead to a mixed shape
- ξ = (1,1,1). Projection of processing domain in parallel of the trihedron bisector lead to a hexagonal shape.

Li and Wah method (Li & Wah, 1984) is very similar to Quinton, the only difference is the use of an algorithm describing a set of uniform recurrent equations giving data spatial distribution, data time propagation and allocation functions for network building.

3.1.2 Mongenet method

The principle of this method lies on 5 steps (Mongenet, 1985) :

- systolic characterization of the problem
- definition of the processing domain
- definition of the generator vectors
- problem representation
- definition of associated systolic nets

3.1.2.1 Systolic characterization of the problem

The statement characterizing a problem must be defined with a system of recurrent equations in $\mathbf{R}^3$:

$$\begin{gathered} y_{ij}^k = f(y_{ij}^{k-1}, a^1, ..., a^n) \\ y_{ij}^k = v,\ v \in \mathbf{R}^3 \\ 0 \leq k \leq b,\ i \in I,\ j \in J \end{gathered} \tag{7}$$

in which $a^1, \ldots, a^u$ are data, I and J are intervals from $\mathbf{Z}$, k being the recurrency index and b the maximal size of the equations system.

a^q elements can belong to a simple sequence (s_l) or to a double sequence ($s_{l,l'}$), $l \in L$, $l' \in L'$, L and L' being intervals of **Z**. In this case, a^q elements are characterized by their indexes which are defined by a function h depending on i, j and k. The result of the probem is a double sequence (r_{ij}), $i \in I$, $j \in J$ where r_{ij} can be defined in two ways :
− the result of a recurrency $r_{ij} = y_{ij}^b$
− $r_{ij} = g(y_{ij}^b, a^1, ..., a^n)$
For example, in the case of resolving a linear equation, results are a simple suite y_i, , $1 \le i \le n$, each y_i being the result of the following recurrency :

$$\begin{gathered} y_i^{k+1} = y_i^k + a_{i,k+1}.\ x_{k+1} \\ y_i^0 = 0 \\ 0 \le k \le n-1,\ 1 \le i \le n \end{gathered} \tag{8}$$

3.1.2.2 Processing domain

The second step of this method consists in determining the processing domain D associated to a given problem. This domain is the set of points with integer coordinates corresponding to elementary processings. It is defined from the equations system defining the problem.
Definition 2. Consider a systolizable problem which recurrent equations are similar to (7) and defined in $\mathbf{R}^3$. The D domain associated to the problem is the union of two subsets D_1 and D_2.:
- D_1 is the set of indexes values defining the recurrent equations system. b being a bound defined by the user, it is defined as $D_1 = \{ (i,j,k) \subset \mathbf{Z}^3, i \in I, j \in J, a \le k \le b\}$
- D_2 is defined as :
  - if the problem result is (r_{ij}) : $i \in I, j \in J \mid r_{ij} = y_{ij}^b$, then $D_2 = \varnothing$
  - if the problem result is (r_{ij}) : $i \in I, j \in J \mid r_{ij} = q(y_{ij}^b, a^1, ..., a^u)$,
    then $D_2 = \{ (i,j,k) \subset \mathbf{Z}^3, i \in I, j \in J, k = b+1 \}$

In the case of the MVP defined in (8), $D_1 = \{ (i,k) \in \mathbf{Z}^2 \mid , 0 \le k \le n-1, 1 \le i \le n\}$ and D_2 is empty, since an elementary result y_i is equal to a recurrency result..
Definition 3. Systolic specification of a defined problem in $\mathbf{R}^3$ from p data families implies that $D \subset \mathbf{Z}^3$ defines the coordinates of elementary processings in the canonical base (b_i, b_j, b_k). For example, concerning the MVP previously defined, $D = \{ (i,k) \in \mathbf{Z}^2 \mid , 0 \le k \le n-1, 1 \le i \le n\}$.

3.1.2.3 Generating vectors

Definition 4. Let's consider a problem defined in $\mathbf{R}^3$ from p data families, and d a data family which associated function h_d is defined in the problem systolic specification.
ψ_d is called a generating vector associated to the d family, when it is a vector of $\mathbf{Z}^3$ which coordinates are (ψ_i ,ψ_j ,ψ_k) in the canonical base BC of the problem, such as :
- for a point (i , j , k) of the D domain, $h_d(i, j, k) = h_d(i+\psi_i , j+\psi_j , k+\psi_k)$
- highest common factor (HCF) is : HCF(ψ_i ,ψ_j ,ψ_k) = +1 or -1

This definition of generating vectors is linked to the fact that (i, j, k) and ($i+\psi_i$, $j+\psi_j$, $k+\psi_k$) points of the domain, use the same occurrence of the d data family.
The choice of ψ_d with coordinates being prime between them enables to limit possible choices for ψ_d and to obtain all points ($i+n x \psi_i$, $j+\psi_j$, $k+\psi_k$), $n \in \mathbf{Z}$, from any (i, j, k) point of D.
In the case of the matrix-vector product, generating vectors $\Psi_y = \Psi_a = \Psi_x = (\psi_y , \psi_a , \psi_x)$ are associated to results h_y, h_a and h_x. Generating vectors are as following :

$h_y(i,k)=h_y(i+\psi_i,\ k+\psi_k) \Leftrightarrow i = i+\psi_i \Leftrightarrow \psi_i = 0$. Moreover, $HCF(\psi_i,\ \psi_k)=\pm1$, thus $\psi_k=\pm1$. Generating vector ψ_y can therefore be (0, 1) or (0, -1).
$h_x(i,k) = i+k$. Generating vector Ψ_a must verify $h_a(i,k)=h_x(i+\psi_i,\ k+\psi_k) \Leftrightarrow i+k=i+k+\psi_i+\psi_k$
$\Leftrightarrow \psi_i = -\psi_k$. Moreover, $HCF(\psi_i,\psi_k)=+1$ or -1, thus $\Psi_a=(1,-1)$ or $(-1,1)$
Similar development leads to $\Psi_x=(1,0)$

3.1.2.4 Problem representation

A representation set is associated to a problem defined in $\mathbf{R}^3$. Each representation defines a scheduling of elementary processings. The temporal order relation between the processing requires the introduction of a time parameter that evolves in parallel to the recurrency, since this relation is a total order on every recurrency processings associated to an elementary processing. We thus call spacetime, the space $ET \subset \mathbf{R}^3$. with orthonormal basis (i, j, t), where t represents the time axis.

Definition 5. A problem representation in ET is given by :
- the transformation matrix P from the processing domain canonical base to the spacetime basis
- the transformation vector V such as $V=\overrightarrow{O'O}$, where O is the origin of the frame associated to the canonical basis and O' is the origin of the spacetime frame

Point coordinates in spacetime can there for be expressed from coordinates in the canonical basis :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{ET} = P.\begin{pmatrix} i \\ j \\ k \end{pmatrix}_{BC} + V$$

$$\Leftrightarrow \begin{pmatrix} i \\ j \\ k \end{pmatrix}_{BC} = P^{-1}.\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{ET} - P^{-1}.V$$

This representation is given by the example of the Matrix Vector Product of Fig. 11.

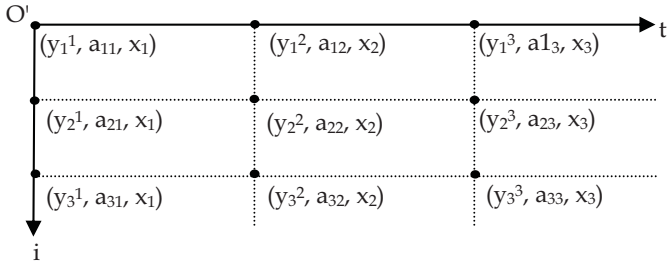


Fig. 11. Representation of the Matrix Vector Product in spacetime (t=k)

We call R_0 the initial representation of a problem, the one for which there is a coincidence between the canonical basis and the spacetime basis, i.e. P = I, I being the Identity Matrix, and V the null vector (O and O' are coinciding). For the MVP example, initial representation is given on Fig. 11.
These representations show the occurencies of a data at successive instants. Processings can be done in the same cell or on adjacent cells. In the first case, data makes a systolic network

made of functional cells in which the data can be put in the cell memory. In the second case, data circulate in the network from cell to cell.
The representation of the problem in spacetime defines a scheduling for the processing. To obtain networks with a different order, we apply transformations on the initial representation R_0. If, after a transformation, data are still processed simultaneously, a new transformation is applied until the creation of an optimal scheduling. From this representation a set of systolic networks is determined.
Applying a transformation to a representation consists in modifying the temporal abscissa of the points. Whatever the representation is, this transformation must not change the n-uple associated to the invariant points when order and simultaneity of processings is changed. The only possible transformations are thus those who move the points from the D domain in parallel to the temporal axis (O', t). For each given representation, D_t is the set of points which have the same temporal abscisse, resulting in segments parallel to (O', i) in spacetime are obtained.
The transformation to be applied consists in deleting data occurencies simultaneities by forcing their successive and regular use in all the processings, which implies that the image of all lines d_t by this transformation is also a line in the image representation. For instance, for the initial representation R_0 of the MVP, D_t straight lines are dotted on Fig. 11. One can therefore see that occurrencies of data x_k, $0 \le k \le n-1$ are simultaneously used on each point of straight line D_k with t = k. Therefore, a transformation can be applied to associate a non parallel straight line to the (O', i) axis to each D_t parallel straight line to (O', i).
Two types of transformations can be distinguished leading to different image straight lines :
- T_c for which the image straight line has a slope = +P (Fig. 12a)
- T_d for which the image straight line has a slope = -P (Fig. 12b)

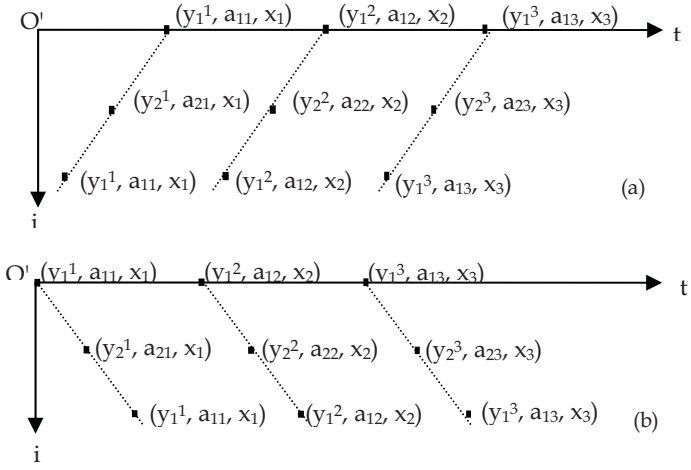


Fig. 12. Applying a transformation on the initial representation : (a) *Tc*, (b) *Td*

The application of a transformation enables to delete the occurencies use simultaneity of data, but increases the processing total execution time. For instance, for the initial representation of Fig. 11, the total execution time is t=n=3 time units, whereas for representations on Fig. 12, it is t=2.n-1 = 5 time units.

Concerning the initial representation, one can notice that 2 points of the straight line D_t having the same temporal abscisse have 2 corresponding points on the image straight line which coordinates differ by 1. It means that two initially simultaneous processings became successive. After the first transformation, no simultaneity in data occurency use is seen, since all elementary processings on D_t parallel to (O', i) use different data. Thus, no other transformation is applied. For the different representations, P (transformation matrices) as well as V (translation vectors) are :

$$P=\begin{pmatrix}1&0\\0&1\end{pmatrix} \text{ and } V=\begin{pmatrix}0\\0\end{pmatrix} \text{ for Fig. 12a}$$

$$P=\begin{pmatrix}1&0\\1&1\end{pmatrix} \text{ and } V=\begin{pmatrix}0\\3\end{pmatrix} \text{ for Fig. 12b}$$

3.1.2.5 Determining systolic networks associated to a representation

For a given representation of a problem, the last step consists in determining what is/are the corresponding systolic network(s). The repartition of processings on each cell of the net has therefore to be carefully chosen depending on different constraints. An allocation direction has thus to be defined, as well as a vector with integer coordinates in $\mathbf{R}^3$, which direction determines the different processings that will be performed in a same cell at consecutive instants. In fact, the direction of allocations can not be chosen orthogonally to the time axis, since in this case, temporal axis of the different processings would be the same, which contradicts the definition.

Consider the problem representation of Fig. 12a. By choosing for instance an allocation direction $\xi=(1, 0)_{BC}$ or $\xi=(1, 1)_{ET}$ and projecting all the processings following this direction (Fig. 13), the result is the systolic network shown on Fig. 14. This network is made of n=3 cells, each performing 3 recurrency steps. The total execution time is therefore $2n$-1 = 5 time units. If an allocation direction colinear to the time axis is chosen, the network shown on Fig. 15 is then obtained.

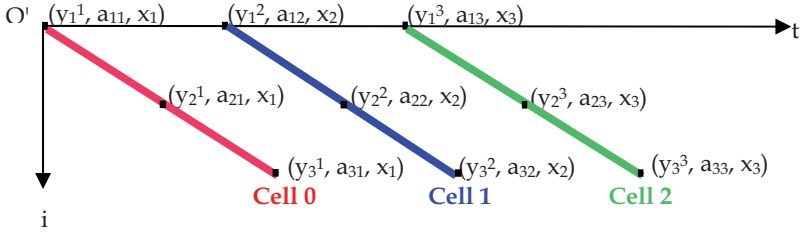


Fig. 13. Processings projection with $\xi=(1,1)_{ET}$

Other networks can be obtained by choosing another value for D_t slope. The nature of the network cells depends on the chosen allocation direction.

Cappello and Steiglitz approach (Capello & Setiglitz, 1983) is close to Mongenet. It differs from the canonical representation obtained by associating a temporal representation indexed on the recurrency definition. Each index is associated to a dimension of the

geometrical space, and each point corresponds to a n-uple of indexes in which recurrency is defined.

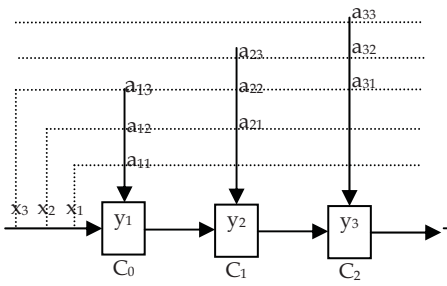


Fig. 14. Systolic network for $\xi=(1,1)_{ET}$

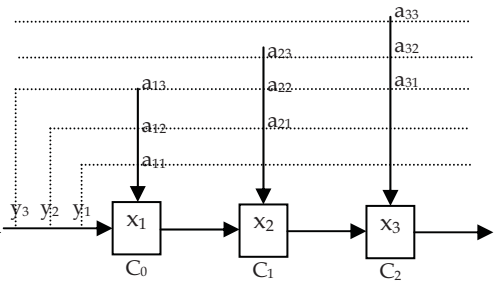


Fig. 15. Systolic network for $\xi=(0,1)_{ET}$

Basic processings are thus directly represented in the functional specifications of the architecture cells. The different geometrical representations and their corresponding architectures are then obtained by applying geometrical transformations to the initial representation.

3.2 Methods using sequential algorithms

Among all methods listed in (Quinton & Robert, 1991), we'll detail a bit more the Moldovan approach (Moldovan, 1982) that is based on a transformation of sequential algorithms in a high-level language.

The first step consists in deleting data diffusion in the algorithms by moving in series data to be diffused. Thus, for (n×n)-matrices product, the sequential algorithm is :

$$\forall i \mid 1\le i\le n, \forall j \mid 1\le j\le n, \forall k \mid 1\le k\le n, c_{new}(i,j)=c_{old}(i,j) + a(i,k).b(k,j) \qquad (9)$$

If one loop index on variables a, b and c is missing, data diffusion become obvious. When pipelining them, corresponding indexes are completed and artificial values are introduced so that each data has only one use. New algorithm then becomes :

$$\forall i \mid 1\le i\le n, \forall j \mid 1\le j\le n, \forall k \mid 1\le k\le n$$
$$a^{j+1}(i, k) = a^{j}(i, k)$$
$$b^{i+1}(k, j) = b^{i}(k, j)$$
$$c^{k+1}(i, j)= c^{k}(i, j)+ a^{j}(i, k).b^{i}(k, j)$$

The algorithm is thus characterized by the set L^n of indexes of n overlapped loops. Here,

$$L^3 = \{ (k,i,j) \mid 1\le k\le n, 1\le i\le n, 1\le j\le n \}$$

which corresponds to the domain associated to the problem.

The second step consists in determining the set of dependency vectors for the algorithm. If an iteration step characterized by a n-uple of indexes $I(t) = \{i^1(t), i^2(t), ..., i^n(t)\} \in L^n$ uses a

data processed by an iteration step characterized by another n-uple of indexes $J(t)= \{ j^1(t), j^2(t), ..., j^n(t) \} \in L^n$, then a dependency vector $DE(t)$ associated to this data is defined :

$$DE(t) = J(t) - I(t)$$

Dependency vectors can be constant or depending of L^n elements. Thus, for the previous algorithm, processed data $c^k(i,j)$ at the step defined by (i, j, k-1) is used at the step (i, j, k). This defines a first dependency vector $d_1=(i, j, k) - (i, j, k-1) = (0, 0, 1)$. In the same way, step (i, j, k) uses the $a^j(i, k)$ data processed at the step (i, j-1, k) as well as the $b^i(j, k)$ data processed at the step (i-1, j, k). The two other dependency vectors of the problem are therefore $de^2=(0,1,0)$ and $de^3=(1,0,0)$.
The next step consists in applying on the $<L^n, E>$ structure a monotonous and bijective transformation T (E is the order imposed by the dependency vectors), defined by :

$$T : <L^n, E> \rightarrow <L_T^n, E_T>$$

T is partitionned into :

$$\Pi : L^n \rightarrow L_T^k, k<n$$
$$S : L^n \rightarrow L_T^{n-k}$$

k gives the dimension of Π and S. It is such as the function results in the order E_T. Thus, the k first coordinates of $\underline{I}$ and L_T^n depend on time, whereas the following n-k coordinates are linked to the algorithm geometrical properties. For obtaining planar results, n-k must be less or equal than 2.
In the case that the algorithm made of n loops is characterized by n constant dependency vectors

$$DE = \{de_1, de_2, ... , de_n\}$$

the transformation T is chosen linear, i.e. $J = T . I$

If v_i is the dependency vector de_j after transformation, $V_i = T. DE_j$, the system to solve is $T.DE = \Delta$, $DE = \{ v_1, v_2, ..., v_m \}$. Necessary and sufficient conditions for existence of a valid transformation T for such an algorithm are :
- $v_i = DE_i[c_j]$, c_j being the HCF of the d_j elements
- $T.DE = \Delta$ has a solution
- The first non-zero element of v_j is positive

Therefore, in our exemple of matrix product, dependency vectors are defined by :

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A linear transformation T is such as $T = \Delta$. The first non-zero element of v_j being positive, we consider $\Pi . d_i > 0$ and $k = 1$ in order to size Π and S, with :

$$T=\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix}$$

In this case, $\Pi.de_i = t_{1i} > 0$. Thus, we choose for $t_{1i,}$ i=1, ..., 3, the lowest positive values, i.e. $t_{11} = t_{12} = t_{13} = 1$. S is determined by taking into account that T is bijective and with a matrix made of integers, i.e. Det(T) = ±1 . Among all possible solutions, we can choose :

$$T=\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This transformation of the indexes set enables to deduce a systolic network :
- Functions processed by the cells are deduced from the algorithm mathematical expressions. An algorithm similar to (9) contains instructions executed for each point of L^n. Cells are thus identical, except for the peripherical ones. When loop processings are too important, the loop is decomposed in several simple loops. The corresponding network therefore requires several different cells.
- The network geometry is deduced from function S. Identification number for each cell is given by $S(I) = (j^{k+1}, ... , j^n)$ for $I \in L^n$. Interconnections between cells are deduced from the n-k last components of each dependency vector v_j after being transformed :

$$v_j^s = S(I + DE_j) - S(I)$$

When T is linear :

$$v_j^s = S.DE_j$$

For each cell, v_j^s vectors indicate the identification number of the cell for the variable associated to the vector. The network temporal processing is given by :

$$\Pi : L^n \rightarrow I_T^k$$

The elementary processing corresponding to $I \in L^n$ is performed at $t=\Pi(I)$. The communication time for a data flow associated to the dependency vector DE_j is given by $\Pi(I+DE_j) - \Pi(I)$, which is reduced to $\Pi(DE_j)$ when T is linear.
Using the integer k for sizes of Π and S with the lowest possible value, the number of parallel operations is increased at the expense of cells number. Thus, when considering the matrix product defined with the following linear transformation :

$$T=\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

S is defined by :