Derivation Of Systolic Algorithms For The Algebraic Path Problem By Recurrence Transformations

Clémentin Tayou Djamegni
Université de Dschang, Faculté des Sciences, Cameroun
dtayou@uycdc.uninet.cm

Patrice Quinton, Sanjay Rajopadhye, Tanguy Risset
IRISA, Campus de Beaulieu, 35042 Rennes-Cedex, France
{quinton, rajopadhye, risset}@irisa.fr

Abstract

In this paper, we are interested in solving the algebraic path problem on regular arrays. We first unify previous contributions with recurrence transformations. Then we propose a new localization technique without long-range communication which leads to a piecewise affine scheduling of $4n + \Theta(1)$ steps, where n is the size of the problem. The derivation of a locally connected space-time minimal solution with respect to the new scheduling constitutes the second contribution of the paper. This new design requires $n^2/3 + \Theta(n)$ elementary processors and solves the problem in $4n+\Theta(1)$ steps, and this includes loading and unloading time. This is an improvement over the best previously known bounds.

Key-words: algebraic path problem, space-time complexity, system of uniform recurrence equations, dependence graph, locally connected graph, localization, timing function, allocation function, regular architectures.

1 Introduction

The algebraic path problem (APP) is an important problem in computer science since it unifies a number of problems arising from various fields: matrix inversion (MI), shortest path between all pairs of a weighted graph (SP), and transitive closure of a binary relation (TC). Let A be an $n \times n$ matrix over a closed semi-ring $S(\oplus, \otimes)$ with a closure operator *. The APP problem is based on the following Affine Recurrence Equations (ARE) [7, 21]:

$$\text{for } 1 \le i,j \le n \;\wedge\; 0 \le k \le n$$
$$f(i,j,k) = \begin{cases} a_{ij} & \text{if } k = 0 \\ f(i,j,k-1)^* & \text{if } k \neq 0 \wedge i = j = k \\ f(i,k,k) \otimes f(i,j,k-1) & \text{if } k \neq 0 \wedge i = k \wedge j \neq k \\ f(i,j,k-1) \otimes f(k,j,k) & \text{if } k \neq 0 \wedge i \neq k \wedge j = k \\ f(i,j,k-1) \oplus f(i,k,k) \otimes f(k,j,k-1) & \text{if } k \neq 0 \wedge i \neq k \wedge j \neq k \end{cases} \tag{1}$$

Following the standard methodology for the systematic synthesis of systolic architectures [14, 16, 17, 19, 20, 26] we first derive a uniform version of (1), which is associated to the Uniform Directed Acyclic Graph (UDAG) $G = (V, U)$ which captures the dependencies between the elementary computations, also called tasks. We then define a timing function (or scheduling) $t : V \longrightarrow N$ which gives the computation date $t(v)$ of each task v of V, assuming that all tasks are of unit delay. The next step is to define an allocation function which assigns the tasks to the processors of a systolic array, i.e., a regular array with local connections. In this approach, the minimum delay necessary to solve the problem is the length of the longest path of G.

The Direct Acyclic Graph (DAG) G associated with the system of equations (1) is not uniform due to non local communications, but its longest path is $3n$ long. In order to obtain a systolic solution to the problem, we uniformize (or localize) the DAG before scheduling and mapping its nodes, so as to synthesize an architecture where communications are local.

The problem of removing non local communications in a more general algorithm such as an ARE has been addressed by a number of authors [17, 19, 20, 30]. There are now well defined methods to systematically transform most AREs into Uniform Recurrence Equations (URE). However the localized version is not unique, and often the dependencies that it introduces add to the critical path of the DAG. The question that arises is whether one can define an uniformization scheme which requires the minimum number of steps.

This question has been studied by several researchers for the particular ARE (1) [7, 21]. These authors have proposed two localization schemes which allow wraparound and lead to piecewise shedulings of $4n+\Theta(1)$ steps. This is a significant improvement over the number of steps, $5n+\Theta(1)$ of schedulings induced by the earlier localization technique. Moreover, it seems that $4n + \Theta(1)$ is the lower bound for the number of steps over all possible nearest neighboring localization schemes for a uniform architecture, but this is not proved either. Rajopadhye [21] proposes a 2D-torus of n^2 cells. The design of Chang and Tsay [7] corresponds to a so-called "spherical array" of n^2 cells. Although the computation time of these arrays is $4n + \Theta(1)$, at least $n/2$ steps are necessary to load the array so that the data $a_{ij}, 1 \leq i, j \leq n$ are available at the beginning, and $n/2$ steps are necessary to unload the array after the computation. Therefore, the loading/unloading operations force a slowdown of their algorithm to the point that $5n+\Theta(1)$ steps are necessary to carry out the APP problem.

An interesting open question is to derive from well known methodologies [5, 14, 16, 17, 19, 20, 26, 30] a local dependence graph which requires the minimum possible number of steps, and an allocation function leading to a space-time minimal systolic array with nearest neighbor connections. It is important to note that this is an intrinsec property of equations (1).

In this paper, we are interested in this last problem. More precisely, we derive a new dependence graph with nearest neighbor connections which leads to a piecewise affine scheduling of $4n + \Theta(1)$ steps. Then, from this new uniformization scheme, we derive a systolic array of $n^2/3+\Theta(n)$ processors which solves the APP problem in $4n+\Theta(1)$ steps. This is an improvement over the best previously known bounds. Note that we make a distinction between nearest-neighbor arrays and local arrays where the interconnections may "jump" over one or more processors (the so called bounded broadcast facility [22, 31]). This often allows a constant factor improvement, and the method can be applied to the array that we present.

The rest of the paper is organized in the following way. In section 2, some basic definitions and notations are introduced. In section 3, we briefly discuss existing localization strategies and architectures by systematically unifying previous results with recurrence transformations. In section 4, we first present a bidirectional localization technique which leads to a piecewise affine scheduling of $4n + \Theta(1)$ steps. Then we present a space-time minimal design. Concluding remarks are given in the last section.

2 Notations And Definitions

We make a distinction between arrays that require wraparound and those that do not. It is well known that an array with wraparound can be laid out with local interconnections but the interconnections are not nearest neighbor but "jump" over alternate processors. As stated earlier, we seek to avoid this.

When designing regular arrays with wrap-around, one may tolerate wrap-around in the localized version of the initial ARE [7, 21]. However, if one is interested only in arrays which are locally connected, the localized version must not admit non-local communication. In this case, the localized version is unidirectional if no pair of dependence vectors are directed in opposite directions, otherwise it is bidirectional.

We now give some definitions and notations related to equations (1). Let D denote the domain of this system of equations. It is easy to see that $D = \{(x, y, z) \in Z^3 \mid 1 \leq x, y \leq n \textit{ and } 0 \leq z \leq n\}$.

Definition 1 *Given an integer h which is strictly positive, the h-th horizontal slice (resp. vertical slice) is the subset of D defined by* $\{(x, y, z) \in D \mid z = h\}$ *(resp.* $\{(x, y, z) \in D \mid x = h\}$*). The (i,j)-th column is the subset of D defined by* $C(i, j) = \{(x, y, z) \in D \mid x = i \textit{ and } y = j\}$

Let us consider equations (1) and the k-th horizontal slice. In this slice, any point depends on (k, k, k), since other points of the same row and column namely (i, k, k) and (k, j, k) directly depend on it, and all other points directly depend on (i, k, k). Depending of this observation, we give two definitions.

Definition 2 *The pivot of an horizontal slice, is the point such that all other points of the horizontal slice depend on it. A source point of the k-th horizontal slice, is either its pivot or any point which directly depends on it, i.e the points* (i, k, k) *and* (k, j, k)*.*

It is important to note that if one re-indexes the points of equations (1), the positions of source points may change. Here, we will consider only re-indexing functions which retain the domain D of equations (1), and maintain any source point in its original horizontal slice.

3 Synthesis Of Previous Solutions

In this section, we survey previous work on systolic algorithms for the APP, and we unify the various known solutions. We first consider solutions based on unidirectional localization (section 3.1), then solutions based on bidirectional localization schemes (section 3.2).

3.1 Solutions Based On Unidirectional Localization

In order to implement the APP problem on an systolic architecture, the program dependencies must be localized. Using standard methodologies, this is achieved by introducing so-called pipelining functions f_1, f_2 and f_3, and rewriting the body of equations (1). The pipelining functions are derived by analyzing the recurrence and determining the set of index points which require identical arguments. Using the localization scheme proposed by Kung, Lo and Lewis [11], the algorithm can be rewritten as follows:

for $1 \leq i, j \leq n \ \wedge \ 0 \leq k \leq n$

$$f(i,j,k) = \begin{cases} a_{ij} & \text{if } k = 0 \\ f(i,j,k-1)* & \text{if } k \neq 0 \wedge i = j = k \\ f_1(i,j,k) \otimes f(i,j,k-1) & \text{if } k \neq 0 \wedge i = k \wedge j \neq k \\ f(i,j,k-1) \otimes f_3(i,j,k) & \text{if } k \neq 0 \wedge i \neq k \wedge j = k \\ f(i,j,k-1) \oplus f_1(i,j,k) \otimes f_2(i,j,k) & \text{if } k \neq 0 \wedge i \neq k \wedge j \neq k \end{cases} \tag{2}$$

where the pipelining functions are defined by:

$$\begin{aligned} f_1(i,j,k) &= \begin{cases} f_1(i, 1 + (j-2) \bmod n, k) & \text{if } j \neq k \\ f(i,j,k) & \text{if } j = k \end{cases} \\ f_2(i,j,k) &= \begin{cases} f_2(1 + (i-2) \bmod n, j, k) & \text{if } i \neq k \\ f(i,j,k-1) & \text{if } i = k \end{cases} \\ f_3(i,j,k) &= \begin{cases} f_3(1 + (i-2) \bmod n, j, k) & \text{if } i \neq k \\ f(i,j,k) & \text{if } i = k \end{cases} \end{aligned}$$

We first prove the following proposition (an equivalent version has been proved in [21]).

Proposition 1 *$T_1(i,j,k) = 3k - 2 + (i-k)(\bmod n) + (j-k)(\bmod n)$ is an optimal scheduling with respect to the above unidirectional localization scheme and it leads to a total time of $5n-4$ time steps.*

Proof Let's first show that there exists a path of length $T_1(i,j,k)$ ending at node (i,j,k) in the dependence graph associated with equations (2). From these equations, node (k,k,k) depends on $(k,k,k-1)$, and $(k,k,k-1)$ depends on $(k,k-1,k-1)$ which in turn depends on $(k-1,k-1,k-1)$. It follows that node (k,k,k) can only be computed three steps after $(k-1,k-1,k-1)$, i.e. it cannot be computed in less than $3k-2$ steps. The first node to be computed on the k-th horizontal slice is its pivot (k,k,k). From the equation of f_3 , there is a path of length $(i-k)(\bmod n)$ from (k,k,k) to (i,k,k) . Finally from the equation of f_1, there is a path of length $(j-k)(\bmod n)$ from (i,k,k) to (i,j,k). Thus node (i,j,k) cannot be computed in less than $T_1(i,j,k)$ steps.

Moreover, for any pair (v_1, v_2) of nodes of the dependence graph, $T_1(v_1) < T_1(v_2)$ holds if v_1 depends on v_2. Thus T_1 is an optimal scheduling function with respect to the above localization scheme. The maximum running time of this localized algorithm is reached for the point $(n-1, n-1, n)$ which is computed at step $5n-4$. $\Box$

This localization scheme allows wraparound. Kung, Lo and Lewis [11] have proved that a starting point to avoid wraparound is by re-indexing the points of equations (1) in such a way that all source nodes are located in the first row or the first column in one horizontal slice. Their re-indexing function is illustrated in figure 1, where the the set of points that serve as source nodes are represented as dark lines. An analytic form of this re-indexing

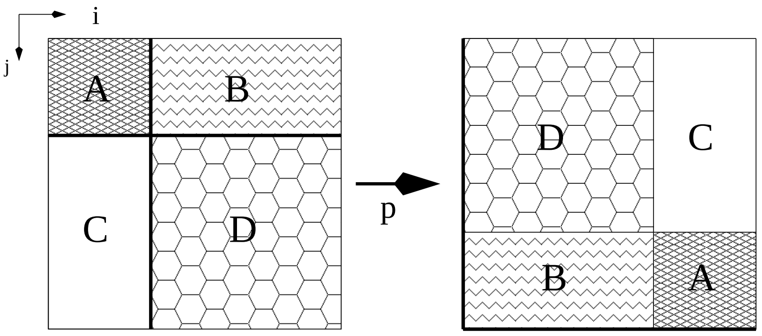

Figure 1: Effect of the p re-indexing function on one horizontal slice

function is given by

$$(i,j,k) \longrightarrow p(i,j,k) = ((i-k)(\bmod n)+1, (j-k)(\bmod n)+1, k) \quad .$$

Clearly, p is one-to-one and its inverse is:

$$p^{-1}(i,j,k) = ((i+k-2)(\bmod n)+1, (j+k-2)(\bmod n)+1, k)$$

For any recurrence equation, it is well known that an equivalent recurrence corresponding to a bijection re-indexation p can be constructed simply by replacing each dependency d by $p \circ d \circ p^{-1}$ and by changing the domain D' of each branch by $p(D')$. This leads to equations (3)

$$f'(i,j,k) = \begin{cases} a_{ij} & \text{if } k=0 \\ f'(i+1,i+1,k-1)^* & \text{if } k \neq 0 \wedge i=1 \wedge j=1 \\ f'(i,1,k) \otimes f'(i+1,\mathrm{rot}(j),k-1) & \text{if } k \neq 0 \wedge i=1 \wedge j \neq 1 \\ f'(\mathrm{rot}(i),j+1,k-1) \otimes f'(1,j,k) & \text{if } k \neq 0 \wedge i \neq 1 \wedge j=1 \\ f'(\mathrm{rot}(i),\mathrm{rot}(j),k-1) \oplus f'(i,1,k) & \text{if } k \neq 0 \wedge i \neq 1 \wedge j \neq 1 \\ \qquad \otimes f'(2,\mathrm{rot}(j),k-1) & \end{cases} \tag{3}$$

where $\mathrm{rot}(i) = i(\bmod n)+1$. If we allow wraparound in the final architecture, we can derive a system of UREs from (3) using standard localization techniques as follows.

$$f'(i,j,k) = \begin{cases} a_{ij} & \text{if } k=0 \\ f'(i+1,j+1,k-1)* & \text{if } k \neq 0 \wedge i=1 \wedge j=1 \\ f_1(i,j,k) \otimes f'(i+1,\mathrm{rot}(j),k-1) & \text{if } k \neq 0 \wedge i=1 \wedge j \neq 1 \\ f'(\mathrm{rot}(i),j+1,k-1) \otimes f_2(i,j,k) & \text{if } k \neq 0 \wedge i \neq 1 \wedge j=1 \\ f'(\mathrm{rot}(i),\mathrm{rot}(j),k-1) \oplus f_1(i,j,k) & \text{if } k \neq 0 \wedge i \neq 1 \wedge j \neq 1 \\ \qquad \otimes f_3(i,j,k) & \end{cases} \tag{4}$$

The three unidirectional pipelining functions f_1, f_2 and f_3 are defined by:

$$f_1(i,j,k) \quad = \quad \begin{cases} f_1(i,j-1,k) & \text{if } 2 \leq j \leq n \\ f'(i,j,k) & \text{if } j=1 \end{cases}$$

$$f_2(i,j,k) = \begin{cases} f_2(i-1,j,k) & \text{if } 2 \le i \le n \\ f'(i,j,k) & \text{if } i = 1 \end{cases}$$

$$f_3(i,j,k) = \begin{cases} f_3(i-1,j,k) & \text{if } 3 \le i \le n \\ f'(i,\mathrm{rot}(j),k-1) & \text{if } i = 2 \end{cases}$$

On the other hand, if one is only interested in nearest neighbor interconnections, wraparound should be removed in equations (4). Since any dependence vector induced by equations (4) can be written as a positive linear integral combination of the vectors $\vec{i}$, $\vec{j}$, $\vec{i}+\vec{j}+\vec{k}$, $-\vec{i}+\vec{j}+\vec{k}$, $\vec{i}-\vec{j}+\vec{k}$, and $-\vec{i}-\vec{j}+\vec{k}$, there exists [17] a uniform version of this equation associated with a UDAG G' which admits these six vectors as dependence vectors. This last uniformization admits $T_2(i,j,k) = i+j+3k-4$, as an optimal timing function which leads to a total time of $5n-4$ steps.

A number of systolic arrays that have been proposed can be derived from G'. In particular, by projecting G' along the $\vec{j}$ axis, we obtain the Robert-Trystram [24], Kung-Lo-Lewis [11], and Lewis-Kung [12] solutions. Three spacetime-minimal solutions have been proposed independently by Benaini-Robert [2], Scheiman-Cappello [4] and Clauss-Mongenet-Perrin [6]. In particular, the Benaini-Robert and Scheiman-Cappello solutions can be obtained by folding the arrays obtained by projecting G' along the $\vec{i}$ and the $\vec{j}$ axes respectively. Assuming that $n = 3m$ for the sake of convenience, these strategies for mapping the set of nodes onto processors correspond to the allocation function $(i,j,k) \longrightarrow (i,(k-1)(\mathrm{mod}\ m)+1)$ in [2], and $(i,j,k) \longrightarrow (j,(k-1)(\mathrm{mod}\ m)+1)$ in [4].

3.2 Solutions Based On Bidirectional Localization

Henceforth, we assume that $n = 2r$ without loss of generality. Note that, if we allow bidirectional localization and wraparound, the maximum number of steps needed to propagate the value of a variable to other points in the same row or column is r. The earliest bidirectional localization scheme based on this observation is due to Rajopadhye [21]. His solution can be obtained by attaching pipelining functions to equation (2) as shown below. The three pipelining functions define four different localization patterns in the four regions delimited by the source nodes.

$$f_1(i,j,k) = \begin{cases} f_1(i,1+(j-2) \bmod n,k) & \text{if } 0 < (j-k) \bmod n \le r \\ f(i,j,k) & \text{if } j = k \\ f_1(i,1+j \bmod n,k) & \text{if } r < (j-k) \bmod n < n \end{cases}$$

$$f_2(i,j,k) = \begin{cases} f_2(1+(i-2) \bmod n,j,k) & \text{if } 0 < (i-k) \bmod n \le r \\ f(i,j,k-1) & \text{if } i = k \\ f_2(1+i \bmod n,j,k) & \text{if } r < (i-k) \bmod n < n \end{cases}$$

$$f_3(i,j,k) = \begin{cases} f_3(1+(i-2) \bmod n,j,k) & \text{if } 0 < (i-k) \bmod n < r \\ f(i,j,k) & \text{if } i = k \\ f_3(1+i \bmod n,j,k) & \text{if } r < (i-k) \bmod n < n \end{cases}$$

Let T_3 be the function defined by $T_3(i,j,k) = 3k-2+t(i,k)+t(j,k)$, where

$$t(i,k) = \begin{cases} (i-k) \bmod n & \text{if } 0 \le (i-k) \bmod n < r \\ 1+(k-i) \bmod n & \text{if } r \le (i-k) \bmod n < n \end{cases}$$

We can prove the following [21]:

Proposition 2 *T_3 is an optimal scheduling with respect to the above bidirectional localization scheme and it leads to a total time of $4n - 4$ time steps.*

The proof is omitted since it is similar to that of proposition 1.

Rajopadhye's systolic array [21] is obtained by computing index (i, j, k) in cell (i, j) for all k. The resulting array is a 2D-torus of n^2 cells. The computation time of this array is $4n - 2$, but it requires r additional steps to load $a_{i,j}$ in cell (i, j) and r steps to unload the array. Thus the total computation time is $5n - 4$ steps.

In [7], Chang and Tsay propose another bidirectional localization scheme which induces a scheduling of $4n + \Theta(1)$ steps. As in the solution of Rajopadhye [21], their approach consists in reducing the number of steps needed to propagate a variable on any horizontal slice from n to $n/2$. However, they proceed in a different way. Indeed, the points of equations (1) are are re-indexed in such a way that all source nodes are located on column number r or on row number r. As in [21], their localization scheme allows wraparound, and their design corresponds to the so-called spherical array (with non-local connections) of n^2 processors which solves the APP problem in $5n + \Theta(1)$ steps due to loading and unloading operations. On the other hand, every processor is active at most one step over three. Then by using a method similar to [3, 4, 6, 22, 27, 28], one can compress their array by a factor of three. This merging of processors reduces the cell count down to $n^2/3$, and it also reduces the loading (resp. unloading) time down to $n/3$ (resp. $n/3$) if it is performed cleverly. Table 3.2 summarize the various solutions to the APP algorithm.

Reference	Application	Area	Time	Period	Load	Unload
Kung, Lo [10]	TC	n^2	$7n$	$3n$	0	0
Rote [25]	APP	n^2	$7n$	$4n$	0	0
Nash, Hansen [15]	MI	$3n^2/2$	$5n$	n	0	0
Guibas et al. [9]	TC	n^2	$6n$	$4n$	n/2	n/2
Robert, Tchuenté [23]	MI	n^2	$5n$	n	0	0
Robert, Trystam [24]	APP	n^2	$5n$	$2n$	0	0
Kung, Lo, Lewis [11]	APP	n^2	$5n$	n	0	0
Lewis, Kung [12]	APP	n^2	$5n$	n	0	0
Benaini et al. [1]	APP	$n^2/2$	$5n$	$4n$	0	0
Benaini et al. [3]	APP	$n^2/2$	$5n$	$2n$	0	0
Clauss et al. [6]	APP	$n^2/3$	$5n$	$3n$	0	0
Benaini, Robert [2]	APP	$n^2/3$	$5n$	$3n$	0	0
Cappello, Scheiman [4]	APP	$n^2/3$	$5n$	$3n$	0	0
Rajopadhye [21]	APP	n^2	$4n$	$4n$	$n/2$	$n/2$
Chang, Tsay [7]	APP	n^2	$4n$	$3n$	$n/2$	$n/2$
Delosme [8]	APP	$5n^2/4$	$4n$	n	0	0

Table 1: Some systolic implementations for APP

4 New Results

As in section 3.2, we assume that $n = 2r$ for sake of convenience.

4.1 A Preprocessing By Re-indexation And Uniformization

Here, we propose a bidirectional localized version of equations (1) which is free of wraparound dependencies, compared to the previous solutions, and leads to a piecewise affine scheduling of $4n + \Theta(1)$ steps. As in the solution of Chang and Tsay [7], we proceed by re-indexing the DAG of equations (3). The idea behind this transformation is to locate the source nodes in the middle of the graph, in a symmetric fashion, and so as to avoid wraparound dependencies, as shown in figure 2. To do so, we use the re-indexing function q defined by:

$$(i, j, k) \longrightarrow q(i, j, k) = (q'(i), q'(j), k),$$

where

$$q'(i) = \begin{cases} r - i + 1 & \text{if } 1 \leq i \leq r \\ i & \text{otherwise} \end{cases}$$

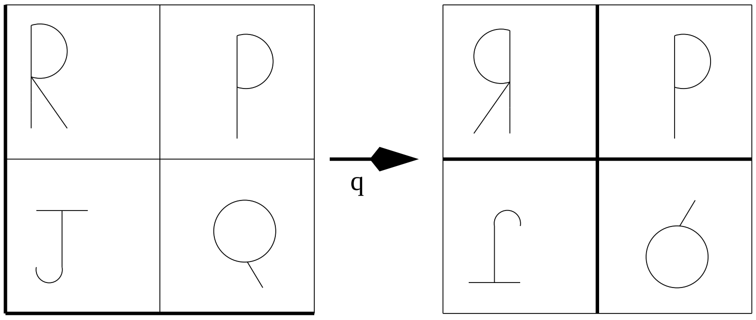

Figure 2: Effect of the q re-indexing function on one horizontal slice of the DAG of (3)

Intuitively (as shown in figure 2 where the source nodes are represented as dark lines), the re-indexing acts differently on the four quarters of any slice specified as:

$$\begin{aligned} J &= \{(i, j, k) \mid 1 \leq i, j \leq r\} \\ R &= \{(i, j, k) \mid 1 \leq i \leq r \ \wedge \ r + 1 \leq j \leq n\} \\ Q &= \{(i, j, k) \mid r + 1 \leq i \leq n \ \wedge \ 1 \leq j \leq r\} \\ P &= \{(i, j, k) \mid r + 1 \leq i, j \leq n\} \end{aligned}$$

The re-indexing q leaves P inchanged, reflects R horizontally, Q vertically and J both horizontally and vertically. Clearly, the function q is one-to-one on the integral cube of size n and is equal to its inverse. By applying this re-indexing function to equations (3), we obtain

$$g(i,j,k) = \begin{cases} a_{q'(i)q'(j)} & \text{if } k = 0 \\ g(i-1, j-1, k-1)^* & \text{if } k \neq 0 \wedge i = r \wedge j = r \\ g(i, r, k) \otimes g(i-1, h(j), k-1) & \text{if } k \neq 0 \wedge i = r \wedge j \neq r \\ g(h(i), j-1, k-1) \otimes g(r, j, k) & \text{if } k \neq 0 \wedge i \neq r \wedge j = r \\ g(h(i), h(j), k-1) \oplus g(i, r, k) & \text{if } k \neq 0 \wedge i \neq r \wedge j \neq r \\ \qquad \otimes\, g(r-1, h(j), k-1) & \end{cases} \quad (5)$$

where

$$h(i) = q'[q'(i)(\text{mod } n) + 1] = \begin{cases} r+1 & \text{if } i = 1 \\ i-1 & \text{if } 2 \leq i \leq r \\ i+1 & \text{if } r < i < n \\ r & \text{if } i = n \end{cases}$$

Equations (5) can be localized by defining bidirectional systolic flows which transport a variable to all points which need it. Let consider two points z and z' of D which are such that z' depends on z according to equations (5). Let define two functions α_i and α_j as follows:

$$\alpha_i(z) = \begin{cases} 1 & \text{if } z \in P \cup Q \\ -1 & \text{if } z \in R \cup J \end{cases}$$

and

$$\alpha_j(z) = \begin{cases} 1 & \text{if } z \in R \cup P \\ -1 & \text{if } z \in J \cup Q \end{cases}$$

Simple linear algebra shows that $z - z'$ is a positive linear integral combination of the three vectors $\alpha_i(z)\vec{i}$, $\alpha_j(z)\vec{j}$ and $-\alpha_i(z)\vec{i} - \alpha_j(z)\vec{j} + \vec{k}$. Thus there exists [17] a uniform version of equations (5) which admits six dependence vectors.

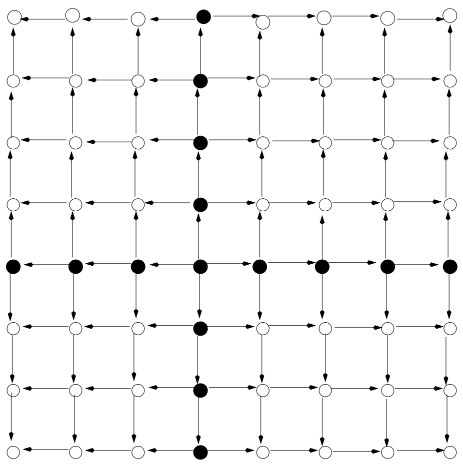

Figure 3: Dependence constraints of G'' on one horizontal slice for $n = 8$

The resulting dependence vectors are: $\vec{i}$, $\vec{j}$, $-\vec{i}$, $-\vec{j}$ inside each slice, and referring to the regions defined in figure 2, $\vec{i} + \vec{j} + \vec{k}$ on region J, $\vec{i} - \vec{j} + \vec{k}$ on region R, $-\vec{i} + \vec{j} + \vec{k}$ on region Q and $-\vec{i} - \vec{j} + \vec{k}$ on region P.

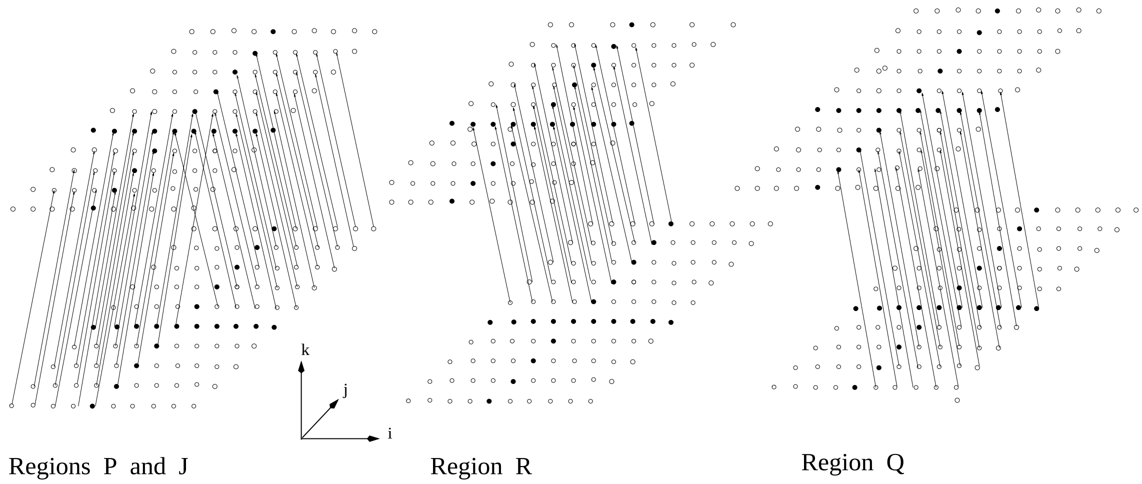

Figure 4: Dependence constraints of G'' between two consecutive horizontal slices for $n = 10$

The dependence vectors between two consecutive horizontal slices are illustrated in figure 4, which shows the four kinds of dependencies in the four regions of a slice. In the following we consider a UDAG G'' associated with such a uniformization.

4.2 A New Timing Function

One can prove the following proposition (the proof is similar to that of proposition 1):

Proposition 3 $T_4(i,j,k) = |i-r|+|j-r|+3k-2$ *is an optimal timing function with respect to the dependences of* G'' *and it leads to a total time of* $4n-2$ *steps.*

Proposition 4 *A lower bound for the number of processors induced by any allocation function which admits* T_4 *as timing function is* $2\lfloor r/3 \rfloor n$.

Proof If we consider three consecutive columns $C(i,j)$, $C(i,j+1)$ and $C(i,j+2)$, then one of these has a calculation scheduled at time $3n-2$. Indeed, $C(i,j)$ is active one tick over three, from step $|i-r|+|j-r|+1$ to $|i-r|+|j-r|+3n-2$, hence in particular at time $3n-2$, one point of these three columns is computed. Now, notice that we can enumerate $2\lfloor r/3 \rfloor n$ such triplets, hence the result. $\Box$

4.3 Limits Of Allocation By Projection

The standard methodology of linear projections is not directly applicable here. Indeed, we cannot choose a projection vector $\vec{p}$ and claim that there are no conflicts simply by checking that $\vec{\lambda}_i.\vec{p} \neq 0$, for the different scheduling vectors $\vec{\lambda}_i$ in each of the regions, as the schedule is not affine. The following proposition show how one can choose an *acceptable projection direction*, that is to say, a projection vector such that two nodes of the dependence graph scheduled at the same time are not mapped on the same processor.

Proposition 5 *A constant unimodular vector* $\vec{p} = (p_1, p_2, p_3)^t$ *is an acceptable projection direction if and only if* $3|p_3| > |p_1| + |p_2|$

Proof
Sufficient condition: We show that, $T_4(u + \vec{p}) \neq T_4(u)$, for all u. Let $u = (u_1, u_2, u_3)$ be a node of the dependence graph. By definition of T_4:

$$\begin{aligned} T_4(u+\vec{p}) - T_4(u) &= |u_1 + p_1 - r| - |u_1 + p_2 - r| + 3(u_3 + p_3) - 2 \\ &\quad -(|u_1 - r| + |u_2 - r| + 3u_3 - 2) \\ &= |u_1 + p_1 - r| - |u_1 - r| + |u_2 + p_2 - r| - |u_2 - r| + 3p_3 \end{aligned}$$

But we observe that:

$$\begin{array}{rcccl} -|p_1| & \leq & |u_1 + p_1 - r| - |u_1 - r| & \leq & |p_1| \\ -|p_2| & \leq & |u_2 + p_2 - r| - |u_2 - r| & \leq & |p_2| \end{array} \tag{6}$$

hence:

$$-|p_1| - |p_2| + 3p_3 \leq T_4(u+\vec{p}) - T_4(u) \leq |p_1| + |p_2| + 3p_3$$

Therefore, since by hypothesis $3|p_3| > |p_1| + |p_2|$, either $T_4(u+\vec{p}) - T_4(u) > 0$ or $T_4(u+\vec{p}) - T_4(u) < 0$.
Necessary condition: Let $u = (r, r, r)$. Assume that $\vec{p}$ be an acceptable projection vector such that $3|p_3| \leq |p_1| + |p_2|$.

- If $|p_1| + |p_2| + 3p_3 = 0$ then $T_4(u) = T_4(u + \vec{p})$. Thus $\vec{p}$ is not an acceptable vector.
- If $-|p_1| - |p_2| + 3p_3 = 0$ then $T_4(u) = T_4(u - \vec{p})$. Thus $\vec{p}$ is not an acceptable vector.
- Suppose that $3|p_3| < |p_1| + |p_2|$. Denote $n_1 = -|p_1| - |p_2| - 3p_3$ and $n_2 = |p_1| + |p_2| - 3p_3$. As $\vec{p}$ is a constant vector, for n big enough, there exists u such that the points $u + n_1\vec{p}$ and $u + n_2\vec{p}$ belong to the graph. In that case, $T_4(u + n_1\vec{p}) = T_4(u + n_2\vec{p})$, which contradicts the fact that $\vec{p}$ is an acceptable vector.

□

From proposition 5, $(0, 0, 1)^t$ is an acceptable projection direction. The corresponding derived array consists of $n \times n$ processors and requires $5n - 2$ time steps since the loading and the unloading of the array require n time steps. However, each processor is active only once every three steps. Thus this array can solve three instances of the problem in $5n + \Theta(1)$ steps. Using a by now standard method [5, 26, 27, 29], the array can be compressed by a factor of 3, thereby reducing the cell count to $n^2/3 + \Theta(n)$. However, this solution is not time-optimal.

Another consequence of proposition 5 is that any acceptable constant projection direction $\vec{p} = (p_1, p_2, p_3)^t$ satisfies $p_3 \neq 0$. Thus the distance between two different processors which are assigned two tasks belonging to the same horizontal slice (in particular the $k = 0$ slice) is at least the distance beetween the two tasks. It follows that arrays obtained by projection require at least r steps for loading so that $a_{i,j}$ is available at the beginning, and at least r steps for unloading since the results are shared by processors at the end of the computation. This remarks force us to use a non linear mapping for obtaining a space-time optimal architecture.

4.4 An Optimal Combinatorial Allocation

We consider here another solution which overcomes this last problem. Our approach can be divided into three phases. Phase 1 leads to a space-minimal solution which solves the problem in $4n+\Theta(1)$ steps, but requires $n/3$ (resp. $n/3$) additional steps for loading (resp. unloading). Phase 2, shows how one can unload the array during the computation at the expense of additional channels. Phase 3, eliminates the additional steps for loading by performing this operation during the computation, and this at this expense of a moderate increase in the bandwidth size of certain channels.

Phase 1 The array induced by this phase is obtained by applying the task allocation heuristic introduced by Salko and Tchuente [26]. Indeed, our starting point is to partition the dependence graph G'' into vertical slices. Then, we seperately allocate the tasks of each vertical slice so as to minimize the processor count. More precisely, the tasks of a vertical slice are recursively allocated by assigning in an alternative manner at each step two columns of $J \cup Q$ and one column of $R \cup P$ or two columns of $R \cup P$ and one column of $J \cup Q$ to a new processor. The target architecture is composed of $\lceil n/3 \rceil$ rows of n cells, numbered (i, j), $1 \le i \le n$, $1 \le j \le \lceil n/3 \rceil$, and the allocation function *alloc* is defined as follows:

$$alloc(i,j,k) = \begin{cases} (i, \left\lceil \frac{2(r-j+1)}{3} \right\rceil) & \text{if } 1 \le j \le r \\ (i, \left\lceil \frac{2(j-r)-1}{3} \right\rceil) & \text{otherwise} \end{cases}$$

Figure 5 shows the allocation on one column of processors (corresponding to the points (i, j, k) where i and k are constant). Let us prove that this allocation is acceptable, that is to say, two different computations are not executed on the same processor at the same time.

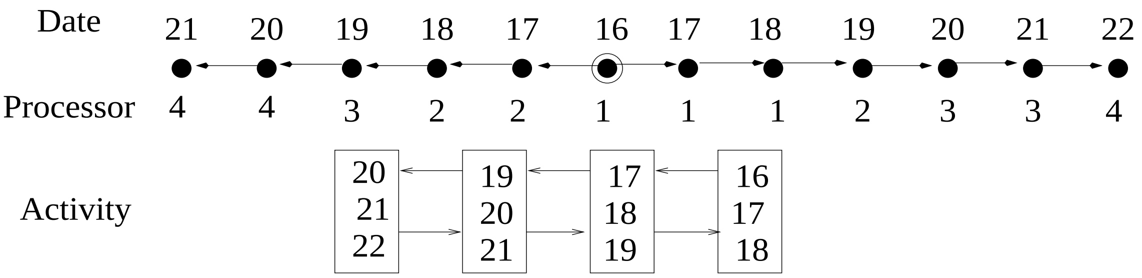

Figure 5: principle of the allocation function *alloc*, shown on a column of processors (represented horizontally) for $n = 12$, $k = 6$.

Proposition 6 *The function alloc is an acceptable allocation function.*

Proof We want to prove that $alloc(i,j,k) = alloc(i',j',k')$ and $(i,j,k) \neq (i',j',k')$ imply that $T_4(i,j,k) \neq T_4(i',j',k')$. If $alloc(i,j,k) = alloc(i',j',k')$, then by definition, $i = i'$. Moreover, we note that if both $1 \le j \le r$ and $1 \le j' \le r$, then $|j' - j| \le 1$. Thus, $|3(k-k')| - 1 \le |T_4(i,j,k) - T_4(i',j',k')| \le |3(k-k')| + 1$, and $T_4(i,j,k) - T_4(i',j',k')$ cannot be equal to 0 if $k \neq k'$. On the other hand, if $k = k'$ then $j \neq j'$ since $(i,j,k) \neq (i',j',k')$ and $(i,k) = (i',k')$. As a consequence, $|T_4(i,j,k) - T_4(i',j',k')| = 1$.

A similar argument holds if $r \leq j \leq n$ and $r < j' \leq n$.

It remains to prove that the property is true when (i, j, k) and (i', j', k') do not lie in the same region, i.e., $1 \leq j \leq r$ and $r < j' \leq n$. We have:

$$T_4(i, j, k) - T_4(i', j', k') = 2r - j - j' + 3(k - k')$$

If $T_4(i, j, k) = T_4(i', j', k')$, then $r - j + 3k = j' - r + 3k'$. Thus $2(r - j + 1) = 2(j' - r) - 1 + (6k' - 6k + 3)$. This implies that

$$\left\lceil \frac{2(r - j + 1)}{3} \right\rceil = \left\lceil \frac{2(j' - r) - 1)}{3} \right\rceil + 2(k' - k) + 1$$

This implies that $alloc(i, j, k) \neq alloc(i', j', k')$. □

The resulting array is space-minimum and performs the APP problem in $4n - 2$ time steps. However, all processors must read some data from outside before processing, and at the end of the computation the results are shared by all processors. Therefore, we must add a first phase to load the network, and after the computation a third phase to unload the network. We present the resulting array in figure 6.

Phase 2 Note that one can carry out the unloading process during the computations at the expense of additional channels. In figure 6, we show the last activation dates of each processor. The additional new connections are shown as dark arrows, one can check that successive processors along this new channel finish with a two step shift, hence there will be no conflict when sending the final coefficient on this channel. With this additional optimization, the execution time with loading is $13n/3 + \Theta(1)$, hence this solution is still not time minimal.

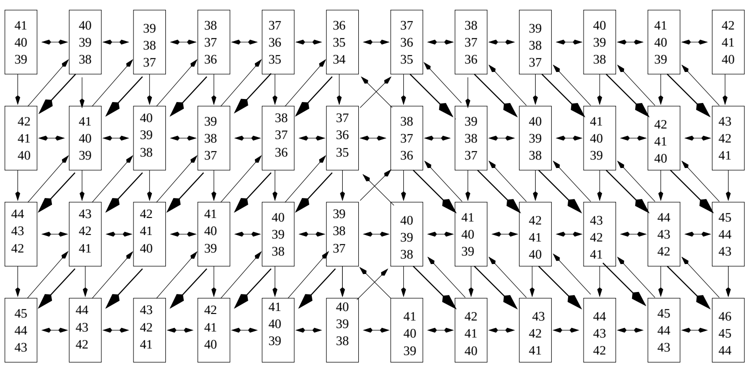

Figure 6: Array obtained with $alloc_1$ for $n = 12$, Channels have been added to unload the array during the computations. The last computation dates for each processors are indicated.

Phase 3 This array can be further optimized in order to perform the loading of the array during the computations. For that, note that, on a given column of processors, the last coefficient entering the array should reach the last row of processors (row number $n/3$) only $n/2$ steps after the first row started working. If we send six coefficients in the array at each cycle, the last coefficient will enter the last row $n/6 + n/3 = n/2$ steps after the first coefficient has been entered. In general the p-th coefficient will reach processor $p/3$ (where it is first used) at time $p/6 + p/3 = p/2$, which is exactely when this processor starts working. Hence by extending the bandwidth of the vertical channel to 6, we can perform the loading of the array during the computations and we obtain a space-time minimal array. This gives a space optimal array which executes the APP in $4n$ steps, which is better than all the results presented in section 3.2.

5 Conclusion

In this paper, we have studied various localization schemes of equations (1). In particular, we have exhibited a bidirectional localization scheme without long-range communication which induces a piecewise affine scheduling of time delay $4n + \Theta(1)$. Then we have derived a locally connected spacetime-minimal systolic arrays (of $n\lceil n/3 \rceil$ processors with respect to our scheduling. These complexities improve the best previously known bounds.

Unifying previous results with recurrence transformations permits to use tools manipulating recurrence equations like the Alpha [13] language and to realize the transformations using implemented function which reduce the error risks. All the re-indexing transformation presented here can be realized by Alpha because they can be decomposed in linear transformations. The problem of implementing the new design presented here under Alpha is a subject of our ongoing research.

Acknowlegements For the first author, this work was supported by the French Agency Aire Développement (Paris, France) through the Project "Calcul Parallèle".

References

[1] A. Benaini, P. Quinton, Y. Robert, Y. Saouter and B. Tourancheau, Synthesis of a new systolic architecture for the algebraic path problem, Science of Computer Programming 15 (1990) 135-158.

[2] A. Benaini and Y. Robert, Space-time minimal systolic arrays for Gaussian elimination and the algebraic path problem, Parallel Comput. 15 (1990) 211-225.

[3] A. Benaini, Y. Robert and B. Tourancheau, A new systolic architecture for the algebraic path problem, In Systolic Array Processors, eds. J. McCanny and al. (Prentice Hall 1989) 73-82.

[4] P. R. Cappello and C. J. Scheiman. A processor-time minimal systolic array for transitive closure. In Application Specific Array Processors, eds. S. Y. Kung and al., IEEE Computer Society Press (1990) 19-30.

[5] P. Clauss, C. Mongenet and G. R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processors arrays, IEEE Internat. Conf. on Application Specific Arrays Processors, ASAP'90 (1990).

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

[7] S. P. Y. Chang and J. C. Tsay, A family of efficient regular arrays for the algebraic path problem, IEEE Trans. Parallel Distrib. Systems, 43 (7) (1994) 769-777.

[8] J. M. Delosme. A parallel algorithm for the algebraic path problem. In Parallel and Distributed algorithm, eds. M. Cosnard et al. (North Holland, 1989), 67-78.

[9] L. Guibas, H. T. Kung and C. D. Thompson. Direct VLSI implementation of combinatorial algorithms. In proc. Caltech Conference on VLSI, Pasadena (1979), 509-525.

[10] S. Y. Kung and S. C. Lo, A spiral systolic algorithm/architecture for transitive closure problems, in Int. Conf. on Circuit Design (ICCD), Rye Town, NY, (1985) 622-626.

[11] S. Y. Kung, S. C. Lo, and P. S. Lewis, Optimal systolic design for the transitive closure and the shortest path problems, IEEE Trans. Comput. C-36 (5) (1987) 603-614.

[12] P. S. Lewis and S. Y. Kung, An optimal systolic array for the algebraic path problem, IEEE Trans. Comput. 40 (1) (1991), 100-105.

[13] C. Mauras. ALPHA: Un lagage équationnel pour la conception et la programmation d'architectures parallèles et synchrones, Ph.D Thesis, Université de Rennes I, IRISA, Campus de Beaulieu, 1990.

[14] D. I. Moldovan, On the analysis and synthesis of VLSI algorithms, IEEE Trans. Comput. C-31 (11) (1982) 1121-1126.

[15] J. G. Nash and S. Hasen. Modified Faddeew algorithm for matrix multiplication. In Proc. SPIE (Society of Photo-Optical Instrumentation Engineers), Real-time Signal Processing (1984), 39-46.

[16] P. Quinton. Automatic synthesis of systolic array from uniform recurrence equations. In Proc. IEEE 11th Annual International Conference on Computer Architecture, Ann Arbor, 1984, 208-214.

[17] P. Quinton and V. Dongen, The mapping of linear equations on regular arrays, J. VLSI Signal Processing 1 (2) (1989) 95-113.

[18] P. Quinton and Y. Robert. Algorithmes et architectures systolique, Masson, 1989.

[19] S. V. Rajopadhye, Synthesizing systolic arrays with control signals from recurrence equations, Distrib. Comput. 3 (1989) 88-105.

[20] S. V. Rajopadhye and R. M. Fujimoto, Synthesizing systolic arrays from recurrence, Parallel Comput. 14 (1990) 163-189.

[21] S. V. Rajopadhye, An improved systolic algorithm for the algebraic path problem, Integration, The VLSI Journal 14 (1993) 279-296.

[22] T. Risset and Y. Robert, Synthesis of processors arrays for the algebraic path problem: unifying old results and deriving new architectures, Parallel Processing Letters 1, (1) (1991) 19-28.

[23] Y. Robert and M. Tchuenté. Résolution systolique de systèmes linéaires denses, RAIRO Modélisation et Analyse Numérique, Technique et Sciences Informatiques 19 (2) (1985) 315-326.

[24] Y. Robert and D. Trystram. Systolic solution of the algebraic path problem. In Systolic Arrays, eds. W. Moore and al., Adam Hilger (1987) 171-180.

[25] G. Rote, A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion), Computing 34 (3) (1985), 191-219.

[26] I. Salko and M. Tchuenté, Une méthode de conception d'algorithmes systoliques pour réseaux réguliers, Technique et Sciences Informatiques 8 (1989) 63-72.

[27] C. T. Djamegni and M. Tchuenté. Scheduling of the DAG associated with pipeline inversion of triangular matrices, Parallel Processing Letters 6 (1) (1996) 13-26.

[28] C. T. Djamegni, Contribution to the synthesis of optimal algorithms for regular arrays, "Thèse de Doctorat", Department of Computer Science, University of Yaounde I, Cameroun, 1997.

[29] M. Tchuenté, Parallel Computation on regular Arrays (Manchester University Press and John Wiley & Sons 1990).

[30] Y. Wong and J. M. Delosme, Transformation of broadcasts into propagations in systolic algorithms, J. Parallel Distrib. Comput. 14 (1992) 121-145.

[31] Yoav Yaacoby and Peter R. Cappello, Bounded broadcast in Systolic Arrays, Technical Report TRCS88-13, Computer Science Department, University of California, Santa Barbara, April 1988.