A Method to Synthesize Modular Systolic Arrays
with Local Broadcast Facility

Tanguy Risset
Laboratoire de l'Informatique du Parallélisme LIP-IMAG
Ecole Normale Supérieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
e-mail: risset@ensl.ens-lyon.fr, risset@frensl61.bitnet

Abstract

In this paper, we propose a method to synthesize modular systolic arrays with local broadcast facility (i.e., arrays containing wires of length lower than a fixed -technology dependent- constant). The synthesis is made from a dependence graph which is not uniform but "locally broadcast". This method aims at generalizing isolated results that have been recently reported on the acceleration of systolic algorithms by using extensions of the "pure" systolic model (wire of length>1, wrap around, folding arrays, ...).

1: Introduction

With the development of VLSI, the systolic model has been much studied as it provides fast implementations of matrix algorithms. Researchers have developed more and more efficient arrays that fit the theoretical systolic model (as given, for example, in [7]). One of the requirement of this original model was that communications between processors should be local, and people have indeed designed arrays where communications only take place between neighbors. The argument for such a restriction is twofold:
(i) modularity: all cells are identical and perform the same computation communication pattern.
(ii) extendability: local communications makes the array directly extendible for larger problems, while non-local communications would imply a re-sizing of transistors and/or signal restoration [10].

Automatic synthesis methods have been developed for this model [3, 6, 8, 11, 12, 13, 14]. Most of these methods start from a "standard" form of the algorithm (linear recurrence equations), that they transform into the well-known uniform recurrence equations (uniformization step). After this step, the uniform dependence graph is scheduled (timing function) and nodes are projected onto physical processors (allocation function) so as to obtain the final systolic array. Both linear and piece-wise linear mappings can be derived in a systematic way, and methods exist to optimize given criteria such as the execution time or the number of processors.

The weakness of the method is precisely the point that makes it feasible: the DAG has to be uniformized before applying space-time transformations (otherwise the problem of

\[^{1}\text{This work has been supported by the Project C3 of the French Council for Research CNRS, and by the ESPRIT Basic Research Action 3280 "NANA" of the European Economic Community}\]
scheduling/mapping a general DAG is very hard, see [6] for more precision on uniformization). However most sequential algorithms lead to a non-uniform DAG, and uniformizing this DAG greatly reduces the space of possible parallel solutions.

Of course local communications are of major importance to VLSI implementations. But implementing pure systolic solutions where communication only occurs between physical neighbors seems to be very restrictive. For instance many linear systolic arrays have been augmented with a broadcast bus for loading/unloading and centralized control, which is a deviation to the philosophy of the model that prevents modularity and extensibility, but which turns out to be very useful in practice. A trade-off would be to allow for a restricted broadcast that would combine all advantages:

* It is still in accordance to the model of uniform DAG.
* It is still feasible for VLSI.
* It is more efficient than pure systolic schemes.

Another motivation for extending the model is the apparition of a generation of distributed memory computers that could be said to "approach" the systolic model (MasPar, Connection Machine, Warp ...). Some of the facilities that they offer (broadcast of data along one direction, longer-than-one communications, ...) cannot be dealt with by current synthesis methods. Hence, we need to extend the mode of possible communications possibilities if we want to synthesize parallel algorithms for such machines.

Recently, several researchers have investigated such an extension [15, 17, 18, 21]. As we lose some regularity with a restricted broadcast extension, the arrays proposed become very complex, hence the need for an automatic correct-by-construction derivation.

It is important to understand that the possibility of such a derivation highly depends upon the structure of non-uniform dependence graphs, and we will precisely define which type of graphs we are able to work with.

The paper is organized as follows: in section 2, we give examples of locally broadcast dependence graphs and of the corresponding systolic arrays, so as to give an intuitive idea of our target architecture. In section 3, we formally define the type of graphs that we consider. In section 4, we present the different steps of the method, explaining them with examples (technical proofs are reported in [19]). In section 5, we give some remarks and complexity results.

2: The intuitive concept of "locally broadcast"

We would like to obtain arrays with wires of length greater than 1 and lower than a constant b. Of course, this transformation is not always possible but in most of the matrix algorithms, there is a broadcast in one direction. For example, in the matrix-matrix product, let index (i,j,k) represent the computation $c_{i,j} = a_{i,k} * b_{j,k}$. Then $a_{i,k}$ is broadcast to the N indices $(i,j,k), 1 \leq j \leq N$ (where N is the matrix-size). On fig1, we have represented the original dependence graph for matrix product (that we could call "globally broadcast graph"); a "locally broadcast graph" we would like to work with, and the well-known uniform graph.
fig1.1: The original graph (N=5)
fig1.2: A locally broadcast graph (N=7)
fig1.3: the uniform graph (N=7)

**Fig1:** Different dependence graphs for matrix product.

We see that the locally broadcast graph is an intermediate form between the uniform graph and the fully broadcast graph. It is not really uniform (all points are not concerned with exactly the same dependence vectors) but it is still modular, while the graph of fig1.1 is not (there exist dependence vectors of length N). There may be numerous locally broadcast graphs for the same algorithm. The graph of fig1.2 can be parametrized by the length of its dependence vectors along each direction (here 3 along i, 2 along j). From the graph of fig1.2, we want to obtain arrays like in fig2.

**Fig2:** Array synthesized from the dependence graph of fig1.2

For this array (fig2), we have used a projection of the locally broadcast graph that could also be applied on the uniform graph (projection along i = (1,0,0): the uniform array obtained is a classical matrix multiplication array [13]), but the timing function is not the same. If you compare the arrays obtained by the same projection from fig1.2 and
fig1.3, we see that the locally broadcast array will go faster (here, \(N\cdot\lceil N/3 \rceil \cdot \lceil N/2 \rceil \) instead of \(3N\)). In this example, we have increased the number of cells by a factor 3 as compared with the classical solution. This is a consequence of choosing \(i = (1,0,0)\) as the projection vector (we will explain this later). We will give other examples where the local broadcast extension provides a faster execution time than the standard solution without any overhead in space.

3: Vocabulary and definitions

\(\mathbb{N}^*\) is the set of positive integers and \(\mathbb{Z}\) is the set of integers. As proposed in [6], we deal with matrix algorithms expressed in terms of a linear recurrence equation of the form

\[
\begin{align*}
\text{where} \\
\begin{array}{l}
ze \mathbb{Z}^n \text{ is an index,} \\
\mathbb{D} \text{ is a set of integral points in a convex polyhedron of } \mathbb{Z}^n (\mathbb{D} \text{ is the domain),} \\
I \text{ is an affine mapping from } \mathbb{Z}^n \text{ to } \mathbb{Z}^n; I = Bz + C, \\
U, V \text{ are variable names,} \\
f \text{ is a function that depends strictly on its arguments.}
\end{array}
\end{align*}
\]

When the index mapping \(I\) reduces to a translation, the equation is said to be uniform.

A dependence vector is the vector \(z-I(z)\) in (LRE): physically, it represents the dependence between the two indices \(I(z)\) and \(z\) (\(I(z)\) must be executed before \(z\)). The dependence graph \(DG=(\mathbb{D},E)\) is the set of indices of \(\mathbb{D}\) linked by the set of all dependence vectors of the system of equations:

\[
E = \{(z_1,z_2) / I(z_2) = z_1 \text{ in (LRE)}\}
\]

To obtain the physical systolic array which executes the algorithm, we apply space-time transformations to the dependence graph. Uniform DAGs are well suited when we restrict to linear transformations (that is: projections of the dependence graph along a time hyperplane and onto a space hyperplane). In this case, conditions that must be satisfied by these transformations are quite simple. In the following, we let \(n = \dim(\text{Vect}(\mathbb{D}))\) and \(N\) will represent the size of the matrix for an execution of a particular algorithm.

Definition: locally broadcast graph.

We say that a dependence graph \(DG=(\mathbb{D},E)\) is locally broadcast along \((v_1,b_1), \ldots, (v_k,b_k)\), where \(v_i \in \mathbb{Z}^n, b_i \in \mathbb{N}^*, 1 \leq i \leq k = \dim(\text{Vect}(\mathbb{D}))\), if there exists a unimodular basis \((v_1,v_2, \ldots, v_k)\) of \(\mathbb{Z}^n\) such that:

\[
\begin{align*}
&v_1, v_2, \ldots, v_k \text{ are dependence vectors (broadcasting vectors)} \\
&\text{Let } z, z_0 \in \mathbb{D} \text{ such that } z-z_0 = \sum_{i=1}^{k} \lambda_i v_i \in \text{Vect}(v_1, \ldots, v_k) \text{ then} \\
&(z_0,z) \in E \iff z = \lambda_0 + (\lambda_1 \mod b_1)v_1 \\
&\text{Let } z, z_0 \in \mathbb{D} \text{ such that } z-z_0 \in \text{Vect}(v_1, \ldots, v_k) \text{ then} \\
&(z_0,z) \in E \iff \{z_0 \in \mathbb{D}, z_1 = (z-z_0) \in \mathbb{D} \implies (z_1,z_0) \in E \}
\end{align*}
\]

\[1: \lambda_i \mod b_i \text{ means } (\lambda_i - 1) \mod b_i + 1\]
This definition ensures that the graph contains only uniform vectors (like \((0,0,1)\) in fig1.2) and "broadcasting vectors" (like \((0,1,0)\) and \((1,0,0)\) in fig1.2). The dependance vectors not contained in \(\text{Vec}(v_1,\ldots,v_k)\) are uniform. \(b_1,\ldots,b_k\) can be seen as the length of the broadcast along \(v_1,\ldots,v_k\).

**4: Description of the method**

The principle is the following: first we derive a regular partition of the graph into "boxes". Then we consider the reduced graph (whose nodes are boxes), and we find a schedule for this new graph which is uniform. Finally we "transfer" the time function back to the original graph and we find an allocation function that cope with this time function.

**4.1: Tiling the domain**

We are working on the graph \(DG=(D,E)\) which is locally broadcast along \((v_1,b_1),\ldots,(v_k,b_k)\) where \((v_1,v_2,\ldots,v_k)\) is a basis of \(\mathbb{Z}^n\) and \(v_1,v_2,\ldots,v_k\) are dependance vectors. Consider the points of the domain of the form:

\[
s(j_1,\ldots,j_n) = \sum_{i=1}^{k} j_i v_i + \sum_{i=k+1}^{k} j_i v_i
\]

We will call these points "vertices" of the domain. The set of points:

\[
B(j_1,\ldots,j_n) = \{ s(j_1,\ldots,j_n) - \sum_{i=1}^{k} \alpha_i v_i / \ 0 \leq \alpha_i < b_i \}
\]

is called the "block" \(B(j_1,\ldots,j_n)\).

It is easy to check that, owing to the locally broadcast structure of \(DG\), we have:

- If \(z_1, z_2 \in B(j_1,\ldots,j_n)\), then \((z_1,z_2)\in E\).
- If \(z = s(j_1,\ldots,j_n) - \sum_{i=1}^{k} \alpha_i v_i / \ 0 \leq \alpha_i < b_i\), then for each \(i\in\{1,k\}\) we have:
  
  if we note \(z_i = s(j_1,\ldots,j_{i-1},j_i-1,\ldots,j_n) - \sum_{i=1}^{k} \alpha_i v_i\), then \((z_i, z)\in E\) (provided that \(z_i\in D\).

This is a technical way to express the following properties (which appear in fig3):

- There is no dependence vector inside a block.
- Each point of a block depends upon a point which is on the frontier of the previous block along each direction \(v_1,\ldots,v_k\).

In figure 3, we have represented the decomposition into blocks corresponding to a locally broadcast graph for the APP algorithm. We use the algorithm proposed by \([1]\), where the domain is \(D=(0,j,k) / 0 \leq j < k \leq 1\). We have only represented the dependence vectors going to and leaving from block \(B(2,2,0)\). The corresponding basis is \(v_1=(1,0,0), v_2=(0,1,0), v_3=(1,1,0)\), and \(v_3=(1,1,1)\). The graph is locally broadcast along \((v_1,3)\) and \((v_2,2)\), while the last dependence vector is \(d_1=(0,0,1)\in \text{Vec}(v_1,v_2)\). We easily check on this example the properties that we have just enumerated.
4.2: Compressing and scheduling the graph

Now that we have decomposed the domain into blocks, we will consider the "compressed dependence graph" (still along (v1,b1),..,(vk,bk)): CDG=(Q,E) where:

Q={B(j1,...,jn) / s(j1,...,jn)∈ D}

E'={(q1,q2) / ∃ z∈ q1 ∃ z∈ q2 / (z1,z2)∈ E}

This graph is the set of vertices of the domain D, linked by the dependence vectors existing between the corresponding blocks. In [19], we prove the following results:

Theorem 1:
if DG is locally broadcast along (v1,b1),..,(vk,bk) and the other uniform dependence vectors being d1,...,dn, then:
- CDG is uniform
- CDG contains less than k+1 2k dependence vectors which can be enumerated by:

CDG={(v1,...,vn)∪(d1/b1),..,(dn/bn)}

[d1/b1] is a notation to enumerate the 2k dependence vectors coming from d1:
if d1=(d11,...,d1m) in the basis (v1,...,vn) then
[d1/b1]=(d11/b11),[d12/b21],...,d1k/bk1,...,d1m/bm1)

where [d1/bj] denotes either [d1j/bj1] or [d1j/bj2].

We give here a sketch of proof, the complete demonstration of the theorem can be found in [19]. One can easily see that the structure of the locally broadcast graph is such that all the non uniform dependence vectors of DG are contained in Vect(v1,...,vn) and are between points of neighbouring blocks (it comes directly from the definition). Thus, when compressing the graph (that is: reducing each block to one point), all the non uniform dependence vectors become uniform.

There is a technical proof of the second point of the theorem in [19]. This point is here to ensure that the uniformisation of the "shape" of the dependence vectors does not imply a too large increase of the number of dependence vectors (even if 2k can be seen as a "large increase", it does not depends upon N-size of the problem).
Theorem 2:
- If we find a linear timing function \( T \) that works for CDG (i.e. that respects the dependence vectors of CDG), then the graph DG will be said to be "implementable" and the timing function \( T \) on DG defined by: 
  \[
  T(z) = T(j_1, \ldots, j_n)
  \]
  does respect the dependence vectors of DG.

This comes from the fact that every dependence vector \( v \) of DG can be written \( v = \lambda u \) where \( u \) is a dependence vector of CDG. Thus if \( T \) respect all the dependence vectors of CDG, \( T \) respect all the dependence vectors of DG. Be aware that \( T \) is not linear.

Finding a linear function for CDG is easy because CDG is uniform (see [13]). This theorem point out the main advantage of this method. As CDG contains less computation points than DG (each "box" of \( b_1 \* \ldots \* b_k \) points is reduced to one point), the execution time of CDG will be, in most cases, much faster than the execution time of the uniform graph obtained from DG (like in [6]). Unfortunately, this is not always true. In section 5, we give a condition that ensure us that working with a locally broadcast graph will accelerate the execution time (compared to the corresponding uniform graph).

In the case of the APP broadcast along \( ((1,0,0),5) \) and \( ((0,1,0),2) \) (cf fig3 and fig4), the compressed dependence graph is composed by the vertices \( s(3,2,k) \) (on the \((v_1,v_2,v_3)\) basis: \( v_1=(1,0,0) \), \( v_2=(0,1,0) \), \( v_3=(1,1,1) \)) linked by the dependences between the blocks they belong to. The new dependence vectors expressed in the basis \((v_1,v_2,v_3)\) are:

\( CDV = \{v_1, v_2, (-1,-1,1), (-1,0,1), (0,-1,1), (0,0,1)\} \).

The new graph and how the new dependence vectors are obtained is shown on fig4. We obtain so many new dependence vectors because the same dependence vector \( d_j \) issued from one block \( B \) can reach several blocks (less than \( 2^k \) because \( d_j \) is uniform), depending on which point it is applied to in the block \( B \).

For example, from block \( B(1,1,0) \), the dependence vector \( d=(0,0,1) \) (on \((i,j,k)\)) can reach four blocks: \( B(0,0,1) \) \( d_1 \), \( B(0,1,1) \) \( d_4 \), \( B(1,0,1) \) \( d_2,d_3 \), and \( B(1,1,1) \) \( d_5,d_6 \) (on \((v_1,v_2,v_3)\)). Blocks and vertices are always express in the \((v_1,v_2,v_3)\) basis. Thus on CDG, \( d \) will generate four dependence vectors: \( d_1 \) (from \( d_1 \)), \( d_2 \) (from \( d_2 \) and \( d_3 \)), \( d_3 \) (from \( d_4 \)) and \( d_4 \) (from \( d_5 \) and \( d_6 \)). We can obtain formally these vectors by applying the enumeration described before, on \( d=(-1,-1,1) \) (on \((v_1,v_2,v_3)\)): \( d_1' = \{(-1,-1,1), (-1,-1,1), (-1,-1,1), (-1,-1,1)\} \).
4.3: Projecting the graph

To get the final architecture, assuming that we have found a timing function, we project the locally broadcast graph $DG$. Unfortunately, as the graph is not uniform and the time function $T$ is not linear, it is very difficult to find conditions that ensure that there will be no collision in the projected graph (in other word, that no two indices will be executed at the same time in the same cell). On the other hand, we know a very simple way to project the compressed graph $CDG$ because it is uniform and $T'$ is linear. To find a correct allocation function for $DG$, we will first find an allocation function $A'$ that works for $CDG$ and we will derive from $A'$ an allocation function $A$ for $DG$.

Suppose that $DG$ is implementable. Then we have a linear timing function $T'$ for $CDG$. We project $CDG$ along one vector of the basis $(v_1, v_2, v_3)$ onto the others. The reason why we force the projection to be done along one of these vectors is the following: we can easily find a condition on the allocation function $A'$ for $CDG$ (see [13]) but we have to provide other conditions if we want $A'$ to fit with $DG$. Indeed, the condition between $A'$ and $T'$ forbid collisions between vertices of the graph $DG$, but collisions between a vertex and a non-vertex can still happen (for instance, if one project the graph of fig.1.2 along vector $(1,1,1)$). To forbid these kinds of collisions, we will impose that, if two blocks have a non-empty projection intersection, then their vertices are projected on the same points. Then a collision between two points of different blocks will imply a collision of the vertices of these blocks which is impossible because it would imply a collision when projecting $CDG$. The case of a collision between two points of the same block can still happen, we will see how to transform the projection to solve this problem. So, projecting along one of the basis vector is not necessary, but is sufficient to ensure that $A$ will not provide collisions between points of different blocks.

We consider two cases:
(a) the projection vector is not parallel to any of the broadcasting vectors,
(b) the projection vector is parallel to a broadcasting vector.

Case (a)
When the projection vector is not a broadcasting vector, we choose $A=A'$ and the projection of the graph $DG$ is classical like in [13]: the cells are given by the computation points and the wires are given by the dependence vectors between these computation points. The number of registers on a link coming from $I$ to $J$ is given by $T(J)-T(I)-1$. As the time function $T$ is not linear, we are not sure that the number of registers will be the same for all projections of the same dependence vector. For example on fig. 3, $T(1,1,0)=1$ (d1) while $T(2,2,1)-T(2,2,0)=2$ (d5) (indices expressed in the $(i,j,k)$ basis) still, $(1,1,0)-(1,1,0)=(2,2,1)-(2,2,0)=(0,0,1)$ (we say that the vectors d1 and d5 are vectors of "different type"). As the number of registers on a wire is different when the wire comes from vectors of different types, we must ensure that two vectors of different types will not be projected between the same points. In [19], we show that, if two points $z_1$ and $z_2$ are projected onto the same point $z$, the corresponding vertices $s_1$ and $s_2$ will be projected on the same point $s$. Moreover, if $z_1=s_1-2\mu_1v_1 \text{ and } s_2=s_2-\mu_2v_1$, then $s_2$ and $s_2-\mu_2v_1$ have the same coordinates relatively to their
vertices. As we also show that the type of dependence vector applied on an index \( z \) depends only upon its coordinates relatively to its vertex, these two results ensure us that if two points \( z_1 \) and \( z_2 \) are projected on the same point \( z \), the type of dependence vector that applies on \( z_1 \) and \( z_2 \) is the same.

Notice that, in this case (a), the array will be composed of "blocks of cells" executing \( b_1 \) computations simultaneously (these blocks of cells correspond exactly to the blocks of the graph).

**Case (b)**

When the projecting vector is a broadcasting vector \( v_1 \), we will still manage to have a "block" of \( b_1 \) cells executing the \( b_1 \) computations of a block of the graph. We project the computations like in the first case. The problem is that, as several points of the same block are projected on the same cell, each cell is supposed to execute \( b_2 \) computations at the same time. Thus we will duplicate each cell (by \( b_2 \) times). As \( b_2 \) does not depend upon \( N \), we can still consider that the array is modular. For a point \( z = \sum_{i=1}^{n} z_i v_i \), instead of considering the projection:

\[
A(z) = \sum_{i=1,j \in [1, \ldots, n]} z_i A(v_i),
\]

we consider:

\[
A(z) = \sum_{i=1,j \in [1, \ldots, n]} z_i A(v_i) + (bz_i + z_i \mod b_j) A(v_j).
\]

The wires of the arrays are obtained like in the first case, by projecting the dependence vectors. Finally, we obtain the following result.

**Theorem 3:**

If the locally broadcast graph is implementable, then we have a timing function \( T \) and an allocation function \( A \) that cope with this timing function. This allocation correspond to a projection along one of the basis vector \( v_1, \ldots, v_n \).

The technical proof is given in [19]. Note that the projection of case (a) give rise to an array where the number of cell does not increase (compared to the projection of the uniform graph), we obtain arrays like in Fig5. The projection of case (b) duplicates each cell, we obtain arrays like in Fig2.

**4.4: Examples**

We give three examples to illustrate the whole procedure.

**Example 1:** projection of the locally broadcast matrix product graph of fig1.2 along \( k \).

For the matrix product (graphs in fig1) we have locally broadcast along \((v_1=(1,0,0),3), (v_2=(0,1,0),2)\). The third vector of the basis is also the third dependence vector: \( v_3=(0,0,1) \). The CDG is \((i,j,k) \mid 0 \leq i \leq 3, 0 \leq j \leq 2, 0 \leq k \leq N \) with the three uniform dependence vectors \((1,0,0), (0,1,0)\) and \((0,0,1)\). The time function of CDG is \( T(i,j,k) = i + j + k \). Thus after having "decompressed" the graph we obtain this time function

\[
T(i,j,k) = k + (i-1)/3 + (j-1)/2.
\]

The allocation function is \( A(i,j,k) = (i,j) \). Here we have no problem neither of collision inside a block nor of projecting different types of vectors (all the \((0,0,1)\) are sent on \((0,0,0)\), this is the "case(a)" projection. The array is represented on fig 5. If the first
efficient is input at $t=0$, the last computation is done at $t=N+[\frac{(N-1)/3+1}{2}]$. As the $c_{ij}$ are computed inside the cell, we still have to output the coefficients if we want to get the results (note that this cannot be done using the local broadcast). If we want to get the results faster, we have to project along another direction in order to have the $c_{ij}$ moving in the array; see example 2.

![Diagram of matrix product](image)

**Fig5:** Locally broadcast array along (1,3), (1,2) of the matrix product projected along $k$.

**Example 2:** Projection of the matrix product locally broadcast graph of fig 1.2 along $j$.

![Diagram of matrix product](image)

**Fig6:** Locally broadcast array along (1,3), (1,2) of the matrix product projected along $j$.

We want the coefficients of $C$ to move in the array, thus $d=(0,0,1)$ must not be projected on $(0,0)$. We project along $j=(0,1,0)$. We are in the case where one of the
broadcasting vector is projected on 0 (case b). Thus, as indicated before, we project the graph (we obtain the bold cells of fig 6) and duplicate \( b_2 - 1 \) times each cell. Then, we put a wire between each original cell and all its copies (wire of type 1 on fig 6 which corresponds to the projection of the broadcasting vector \( j = (0,1,0) \)). We put the other wires (wires 2 on fig 6) respecting the number of registers (here, none on any wire). To respect exactly the expression of \( A \) given in section 4.3, be really strict, we should have represented the duplicated cells on the same horizontal line as the cells which they come from. Note that the array given in fig 2 is the same locally broadcast DG projected along \( i = (1,0,0) \).

Example 3: projection of the APP locally broadcast graph.

In this last example, we see how we treat the problem of the different types of dependence vectors. We have seen on fig 3 and fig 4 the structure of the APP locally broadcast dependence graph along \((v_1 = (0,0,1), 3), (v_2 = (0,0,1), 2)\) with \( v_3 = (1,1,1) \). The compressed graph is represented on fig 4 and we have seen that the dependence vector \( d_3 = (-1, -1, 1) \) gives four types of dependence vectors on CDG: \((-1,-1,1), (-1,0,1), (0,-1,1)\) and \((0,0,1)\) on \((v_1,v_2,v_3)\) which are \(d_1,d_2,d_3,d_4\) on fig 5. The broadcasting dependence vectors are \((1,0,0)\) and \((0,1,0)\). The optimal timing function of CDG is \( T(i_1,i_2,i_3) = i_1 + i_2 + i_3 \). The projection along \( j \) i.e. \( A(i_1,i_2,i_3) = (i_1,i_3) \) fits with this timing function. We note that \( T(d_1) = 1, T(d_2) = 2, T(d_3) = 2 \) and \( T(d_4) = 3 \), thus the dependence vector \( d_1 \) will be projected on a wire without register, \( d_2, d_3 \) and \( d_4 \) which give \( d_2' \) and \( d_3' \) on CDG will be projected on a wire with one register, and \( d_4' \) will be projected on a wire with two registers. In [19] we explain how to recognize which index is concerned with which type of vector. For example, if we look at vector \( d_1' = (-1,-1,1) = (d_1,d_1',d_2,d_2',d_3,d_3') \), we know that the vectors of this type are applied from points \( z \) such that, there exist a vertex \( s(l_1,l_2,l_3) \) such that \( z=(l_1,l_2,l_3)-(z_1,z_2,z_3) \) with \( b_1(z_1 mod + b_1) + (d_1 mod + b_1) < 2b_1 \) and \( b_2(z_2 mod + b_2) + (d_1 mod + b_2) < 2b_2 \). As \( d_1 = d_2 = 1, d_1 mod + b_1 = d_2 mod + b_2 = 1, \) thus we must have:

\[
3S(z_1 mod + 1) + 6 \leq 0 \leq 3
\]
Thus, between the projections of the two points of the form \( z=(3\lambda_1-2,2\lambda_2-1,\lambda_3) \) and \( z+d_3=(3\lambda_1-1,2\lambda_2-1,\lambda_3+1) \) there will be a wire with no register (as \( T(d'_1)=1 \)).

The projection \( \Delta' \) extended to DG becomes \( A'(i_1,i_2,i_3)=(2i_1+2i_2,modz,i_3) \) (because the projection is parallel to the hadcasting vector \((0,1,0)\)). Thus, between the cells \((6k_1-3,\lambda_3) \) and \((6k_1-1)\lambda_3+1) \), there is a wire with no register (wire 1 on fig7). By applying the same method to \( d'_2 \) (3 on fig7), \( d'_3 \) (2 on fig7), and \( d'_4 \) (4 on fig7), we obtain all the wires coming from \( d_3 \).

5: Results and concluding remarks

In this paper, we have not fully solved the problem of determining the class of dependence graphs where the local broadcast can be useful (that is: where the locally broadcast graph has a timing function that is faster than the uniform graph). This must be treated when generating locally broadcast graphs. From our definition, we cannot assume that the optimal locally broadcast array will go faster than the optimal uniform array. On fig8, we have represented a graph on which the local broadcast along (1,0) can be applied but the resulting locally broadcast graph is slower than the uniform graph. In fact, on this example, the locally broadcast graph and the uniform graph have the same optimal time function: \( T(i,j)=i \). Thus it is impossible to accelerate the uniform graph with the present method.

However, we point out that, if the domain of an algorithm is included in a cube of size \( N_1*N_2*...*N_n \) and if the optimal time function of the uniform graph works on CDG (this is the case in fig1 and also each time the dependence vectors \( d_1,...,d_k \) are basis vectors), then, if we denote \( t(N_1,N_2,...,N_k) \) the optimal execution time of the uniform graph, the execution time of the locally broadcast graph will be less than \( t(N_1/b_1,N_2/b_2,...,N_k/b_k,N_{k+1},...,N_n) \). As soon as \( t \) depends strictly upon one of its \( k \) first arguments, the local broadcast accelerates the execution.

We can also compare performances of arrays obtained on particular algorithms. We have seen that the matrix product can be executed (cf Example 1) in time \( N4[1N/b_1] \) \[4[1N/b_1] \) with \( N^2 \) cells, with the C-coefficients computed inside the cells. Note that this result can be consider as space-time-optimal for a \( b_1*b_2 \) broadcast. Indeed, if we consider the extreme cases: \( b_1=b_2=N \) (total broadcast equivalent to the original graph) or \( b_1=b_2=1 \) (uniform graph), in both cases we obtain time optimal arrays (and also space optimal in the first case). Of course, we still have to output the coefficients, thus the latency between two successive executions on the same array will be \( N \) (at most). If, like in Example 2, we project the graph along \( j \), we obtain the same execution time (which is still time-
optimal) but with $b^*n^2$ cells. Here, we cannot say that the algorithm is space optimal but there is no latency and successive matrix multiplications can be pipelined.

Concerning the APP, we have retrieved the array given in [18] (in this table, to obtain an array with $n^2$ cells, we have also folded the array like in [18]):

<table>
<thead>
<tr>
<th>Reference</th>
<th>Application</th>
<th>Area</th>
<th>Time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rote [16]</td>
<td>general APP</td>
<td>$n^2$</td>
<td>7n</td>
<td>3n</td>
</tr>
<tr>
<td>Robert-Trystram [20]</td>
<td>general APP</td>
<td>$n^2$</td>
<td>5n</td>
<td>2n</td>
</tr>
<tr>
<td>Benaini et al. [2]</td>
<td>general APP</td>
<td>$n^2/2$</td>
<td>5n</td>
<td>2n</td>
</tr>
<tr>
<td>Cappello-Scheiman [4]</td>
<td>general APP</td>
<td>$n^2/3$</td>
<td>5n</td>
<td>3n</td>
</tr>
<tr>
<td>Benaini-Robert [1]</td>
<td>general APP</td>
<td>$n^2/3$</td>
<td>5n</td>
<td>3n</td>
</tr>
<tr>
<td>Lewis-Kung [9]</td>
<td>general APP</td>
<td>$n^2$</td>
<td>5n</td>
<td>n</td>
</tr>
<tr>
<td>Rajopadhye [15]</td>
<td>general APP</td>
<td>$n^2$</td>
<td>5n</td>
<td>5n</td>
</tr>
<tr>
<td>Diegser [5]</td>
<td>general APP</td>
<td>$1.25n^2$</td>
<td>4n</td>
<td>n</td>
</tr>
<tr>
<td>This paper</td>
<td>general APP</td>
<td>$n^2$</td>
<td>$3n+n+b+n/c$</td>
<td>$3n$</td>
</tr>
</tbody>
</table>

The local broadcast method unifies several ways of considering semi-systolic arrays. For example in [15], the way proposed to accelerate the APP (where the coefficients are computed inside the cells) is to duplicate the pivot, and to send it in the four directions with both horizontal and vertical wrap around of the array. This corresponds to a 2-broadcast on both directions $i$ and $j$. Note that the fact of accelerating an array by increasing the length of the wires or by duplicating cells come from the same structure of the dependence graph: the local broadcast along one direction.

Our method still has several restrictions, first on the type of graph that we project, second on the kind of projection that we use, but the advantage of using such a method is that all the processing is automatic, thus we can generate arrays that are correct-by-construction. We have demonstrated that the local broadcast can accelerate the execution of several matrix algorithms. Further work will provide automatic generation of locally broadcast graph and to implement this method on existing machines (like MasPar) where the local broadcast facilities are available.

Acknowledgements

The author would like to thank Alain Darte and Yves Robert for their helpful comments and suggestions.

References


