Structured Scheduling of Recurrence Equations

Tanguy Risset and Patrice Quinton

Irisa, Campus de Beaulieu, 35042 Rennes Cedex, France
{risset,quinton}@irisa.fr

Abstract. We study scheduling of structured systems of recurrence equations. We introduce the notion of structured dependence graph and structured scheduling. We give an algorithm for the computation of linear and multi-dimensional structured scheduling using the methods for computing scheduling of non-structured systems of affine recurrence equations. This work enables us to schedule large systems of affine recurrence equations. We show the efficiency of the structured scheduling by scheduling a structured system which express the singular value decomposition of a matrix.

Keyword: cyclic scheduling, structured recurrence equations, regular parallelism, loop nest, Automatic synthesis methodology, vlsi

1 Introduction

Recurrence equations were introduced as a formalism for describing computations by Karp, Miller and Winograd [17]. This formalism has been studied in the context of the automatic parallelization of programs [19, 15, 28, 6] and of the synthesis of regular arrays [18, 22, 25, 24].

Research on recurrence equations has lead to the definition of various languages such as Lucid [1], Lustre [3], Signal [21], Crystal [4], Pei [26] and Alpha [23].

The scheduling of recurrence equations has been extensively studied [27, 5, 25, 15, 14] and ***** have been used in many parallelizing compilers (APE,PIPS,PAF,LOOPO) and also (more recently) in some high level synthesis tools for more specific designs (Sys3, Arrest, Cathedral IV, HiFi, MMAlpha).*

Today's main application of these researches is the cyclic scheduling (or parallelization) of loops. However, this technique is limited to loops with static control embedded in a single program. Static control loops are... Current research is now focusing on the extension of scheduling methods to non-static control programs or inter-procedural analysis, as well as on the refinement of existing methods for more efficient implementations.**

This paper is concerned with the scheduling of *structured* systems of affine recurrence equations (structured SAREs), hence it can be seen as an extension of the scheduling of SAREs to the scheduling of several SAREs which call each others.

Primarily addressing the need for program structuring, the recent extension of the SARE formalism to structured SAREs [9] allows one to write and manipulate large SARE specifications in a structured form, but also raises several new questions. For instance, when a SARE is used in several different contexts, it is not always possible to define a single schedule of this SARE suitable for all these contexts. For this reason, the scheduling of structured SAREs cannot be directly deduced from the scheduling techniques dealing with unstructured SAREs.

Throughout this paper, we will express systems of recurrence equations in the Alpha language, a precise description of which can be found in [13, ?].

The paper is organised as follows: after a brief review of the existing results on scheduling for recurrence equations, we introduce the notion of structured dependence graph as an extension of the classical dependence graph. Section 4 addresses linear scheduling, it provides a systematic method for computing a structured linear scheduling for a structured SARE. Then, in section 5, this method is extended to a strategy for finding multi-dimensional structured scheduling. The particularity of the methods proposed here is that they use classical scheduling techniques (resolution of a linear programming problem) and hence can be directly implemented on top of a tool for the scheduling of SAREs. Finally, section 6 is an extensive example: we schedule a structured SARE for the computation of

* Reference

** Reference

the singular value decomposition of a matrix, and we discuss various scheduling strategies.

2 Background an definitions

In this section, we introduce the main notions and definitions needed to formalize the problem: affine recurrence equations, Alpha programs, scheduling, reduced dependence graph, and the extensions of these notions to structured systems of recurrence equations.

2.1 Affine recurrence equations

A System of Affine Recurrence Equation (SARE), is a finite number of equations of the form:

$$z \in \mathcal{D} \quad : \quad V_0(z) = f(V_0(I_0(z)), \ldots, V_p(I_p(z)))$$

where:

1. D is the *domain* of the equation: it is the set of points with integral coordinates included in a convex domain of Z^n.
2. n is the *dimension* of variable V_0.
3. For all i, I_i, the *dependency function*, is an integral affine function from Z^n to Z^p, where p is the dimension of variable V_i.

For example

$$\{k \mid k = 0\} : \texttt{Acc}(k) = 0 \tag{1}$$

$$\{k \mid 1 < k \leq N\} : \texttt{Acc}(k) = \texttt{Acc}(k-1) + \texttt{V1}(k) * \texttt{V2}(k) \tag{2}$$

is a SARE which computes the dot product of two N-vectors `V1` and `V2`.

For a given variable V, we denote as $\mathrm{Dom}(V)$ the definition domain of this variable.

2.2 Alpha systems

The Alpha language [23] allows SAREs to be expressed. Fig. 1 show an Alpha program for the dot product of equations (2), with the addition of a variable `res` which isolates the result. In the rest of this paper, we will describe SAREs as Alpha systems.

2.3 Scheduling SAREs

Scheduling a SARE aims at assigning an execution date to the computations specified by the system. As there are usually a very large number of such computations, we look for a closed form in order to express the schedule of a possibly infinite number of computations in a finite manner.

```
system dot : {N | N >= 2}
  (V1,V2 : {k | 1<=k<=N} of integer)
returns (res:  integer);
var
  Acc : {k | 0<= k <= N } of integer;
let
  Acc[k] =
     case
        {| k = 0 } : 0[];
        {| k > 0 } : Acc[k-1]+V1[k]*V2[k];
     esac;
  res[] = Acc[N];
 tel;
```

Fig. 1. Alpha program for the dot product of two vectors

The formalism of recurrence equations naturally leads to the notion of *linear scheduling*: for each variable V of the system, the computation date is expressed as a linear function $T_{\mathtt{V}}(z) = \tau_{\mathtt{V}} \cdot z + \alpha_{\mathtt{V}}$ of its indexes. A linear schedule is valid if it is positive and respects dependencies between the computations. For instance, in the SARE of equation (2), $\mathtt{Acc}(k)$ depends on $\mathtt{Acc}(k-1)$ for all k in $\{k \mid 1 < k \leq N\}$, hence a valid schedule $T_{\mathtt{Acc}}$ must respect the constraint $T_{\mathtt{Acc}}(k) - T_{\mathtt{Acc}}(k-1) > 0$. Therefore, $T_{\mathtt{Acc}}(k) = k$ is a valid linear schedule for this SARE.

2.4 Reduced dependence graph

Cyclic scheduling techniques base their analysis on the data dependencies. For a SARE, the data dependency information is usually represented in the form of a (reduced) dependence graph.

Definition 1. *((reduced) dependence graph) The dependence graph of an Alpha system is a graph $G = (V, E)$ whose vertices V are Alpha variables and edges E represent dependencies between variables. Vertices are labeled by the variables names and domains, and edges are labeled by the dependency function and the definition domain of this function.*

As an example, Fig. 2 shows the DG of the Alpha system of Fig. 1.

2.5 Structured systems of recurrence equations

The Alpha language provides a means to describe structured systems of affine recurrences equations [9], SSARE in the following. For instance, Fig. 3 shows the definition of a matrix multiplication MM by means of N^2 dot products. The third equation, called a `use` statement, instanciates N^2 times the dot product for all

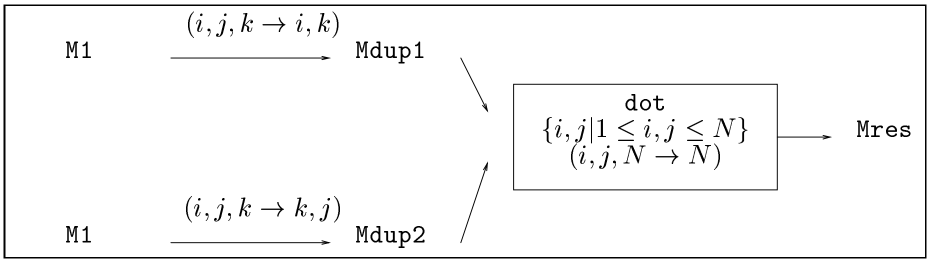

Fig. 2. Dependence graph of the Alpha system for the dot product

```
system MM : {N | N >= 2}
  (M1,M2 : {i,j | 1<= i,j <= N} of integer)
returns (Mres : {i,j | 1<= i,j <= N} of integer);
var
  Mdup1,Mdup2: {k,i,j | 1<= i,j,k <= N} of integer;
let
  Mdup1[k,i,j]=M1[i,k];
  Mdup2[k,i,j]=M2[k,j];
  use {i,j| 1<= i,j <= N} dot[N] (Mdup1,Mdup2) returns (Mres);
tel;
```

Fig. 3. Structured Alpha specification of the matrix product using N^2 dot products.

points (i, j) such that $1 \leq i, j \leq N$. The general form of a use statement in Alpha is the following:

$$\texttt{use } \{J | AJ \leq b\} \texttt{ sub[}f(J, N)\texttt{]} \ (U_1, \ldots, U_n) \texttt{ return } (V_1, \ldots, V_m) \qquad (3)$$

This statement represents a multiple instanciation of the `sub` Alpha system, and it is made out of the following elements:

- The domain $\mathcal{E} = \{J | AJ \leq b\}$ is called the *extension domain* of the statement. It represents the instances of `sub` used. Note on Fig. 3 that, as the formal input of `dot` is one-dimensional and $\mathcal{E} = \{i, j \mid 1 \leq i, j \leq N\}$ is two-dimensional, the actual input of the use of `dot` is three-dimensional. In system `MM`, i and j are *extension indices*.
- The function $f(J, N)$, called the *parameter assignment* function, gives the value of the size parameters of the called subsystem `sub` in term of the parameters of the caller system. In the system `MM` of Figure 3, the parameter assignment function $f(i, j, N) = N$ simply transmits the size parameter N from `MM` to `dot`.

2.6 Structured scheduling

As the works on scheduling recurrence equations have always focused on unstructured recurrence equations, we have to give a formal definition of a *structured* *scheduling*. The notion that we introduce here allows one to reuse components of libraries of (already scheduled) systems.

Definition 2. *(*structured scheduling*) Consider an Alpha system* `sys`*, and a schedule* T^{sys} *of* `sys`*. Let*

$$\texttt{use}\ \{J|AJ \leq b\}\ \texttt{sub[}f(J,N)\texttt{]}\ (U_1, \ldots, U_n)\ \texttt{return}\ (V_1, \ldots, V_m) \qquad (4)$$

be a call statement in `sys`*, to a subsystem* `sub`*. Denote by u_i (resp. v_i) the formal input (resp. output) variable of* `sub` *corresponding to U_i (resp. v_i). We say that* T^{sys} *is* structured *with respect to this call statement if, there exists a schedule* T^{sub} *of* `sub`*, and a linear function $c(J)$ such that:*

$$\forall U_i, \forall (I,J) \in \mathrm{Dom}(U_i),\ T^{sys}_{U_i}(I,J) = c(J) + T^{sub}_{u_i}(I)$$
$$\forall V_i, \forall (I,J) \in \mathrm{Dom}(V_i),\ T^{sys}_{V_i}(I,J) = c(J) + T^{sub}_{v_i}(I)$$

We say that T^{sys} *is a structured scheduled if it is structured with respect to all call statements in* `sys`*.*

$\star\star\star$

2.7 Structured dependence graph

We propose now an extension of the notion of dependence graph to structured SAREs.

Definition 3. *(structured dependence graph,* SDG*) The structured dependence graph of a structured Alpha system is a graph $G = (V,E)$ whose vertices V are labeled by the variables of the system and their domains, and edges E represent either:*

1. *dependencies between variables, these edges being labeled by the dependency function and the domain on which it applies;*
2. *dependencies between actual inputs and actual outputs of a use statements, these edges being labeled with the name of the instanciated system, the extension domain and the parameter assignment function.*

The structured dependence graph of the system of Fig. 3 is shown in Fig. 4. † Note that, unlike the dependance graph, the structured dependance‡ graph does not represent all the information contained in the program. For example, in the SDG of Fig. 4, the dependency between inputs `Mdup1`, `Mdup2` and output `Mres` is not explicitely expressed. This is fair because this information could only be extracted by analyzing the body of the `dot` system.

$\star\star\star$ Give an illustration

† Expliquer

‡ Check spelling

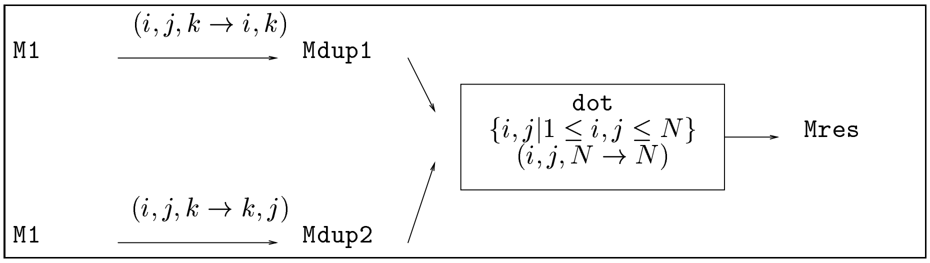

Fig. 4. Structured dependence graph of the Matrix product `MM` of program 3

3 The complexity of scheduling SAREs

In this section, we summarize the principles of finding a schedule for a SARE, and we describe the complexity, both theoretical and practical, of this problem.

3.1 Method

The main difficulty of is that it deals with large or even infinite iteration spaces. This problem was addressed in two contexts: the synthesis of systolic arrays [22, 25, 24] and the automatic parallelization of programs [15, 14, 28, 6, 7] Scheduling uniform recurrences was first considered by Karp, Miller and Winograd [17]. Later on, extensions to linear recurrences [15, 5, 24] were examined. All these methods are based on results of linear programming. For instance, the method depicted in [15] uses the affine form of Farkas Lemma, and the one of [5] uses the duality theorem of linear programming.

In all cases, the problem amounts to solve an integer linear program (ILP), obtained in the following way.

Consider a recurrence equation

$$z \in \mathcal{D} \quad : \quad V_0(z) = f(V_0(I_0(z)), \ldots, V_p(I_p(z))) \quad . \tag{5}$$

To find out a linear schedule T_{V_0}, two types of constraints should be met :

- first, T_{V_0} must be positive on the domain D. As D is a convex polyhedron, this property amounts to satisfy a finite set of linear inequalities, whose variables are the coefficients of the T_{V_0} function, and whose coefficients are obtained using the vertices and rays of D (recall that a *vertex* of a convex polyhedron P is an extreme point of P, and a *ray* is an infinite direction of P.)
- second, for all pair of dependent variables (V_0, V_j) in equation 5, the schedule T_{V_0} and T_{V_j} must be such that $T_{V_0}(z) \geq T_{V_j}(I_j(z))$, for all point $z \in \mathcal{D}$. Again, this property is satisfied if a set of linear inequalities whose variables are the coefficients of T_{V_0} and T_{V_j} are met.

Finding out a schedule amounts therefore to build the system of linear inequalities, and solve it, using ILP. In practice, one try to minimize the total time of the computation for a system, and this is obtained by a lexicographic optimization of the coefficients of the functions.

3.2 Complexity of scheduling

To solve the ILP, one uses methods based on the simplex algorithm. The number of pivoting steps of the simplex method is known to be in practice linear in the number of variables (or linear contraints) of the problem [**?**]. To have a more precise idea of this complexity, we have run this method on a set of Alpha systems.

Table **??** gives the list of systems considered, and the number of vertices, constraints, and total scheduling time. These programs were run on a Sun Workstation... Fig. **??** shows a plot of the number of contraints in term of the number of variables, and shows that there is a very strong linear relationship between these variables. Fig. **??** then gives a plot of the scheduling time in term of the number of variables. This curve was fitted to a cubic expression with a very high significance.

The conclusions of this experiment are three-fold:

1. First, the practical complexity of the scheduling fits near perfectly to that of the simplex algorithm. This shows that the problem has a very good behaviour, as ILP is not necessarily well behaved.
2. Second, the $0(n^3)$ behaviour is a very strong incentive to find out structured methods for solving the scheduling of systems, as we may save in practice a lot of complexity.
3. Finally, we can see that the scheduling time is reasonable for small systems, but becomes prohibitive in the context of automatic design of circuits for large systems.

4 Linear scheduling of structured SAREs

[§] In order to find out a schedule for a structured system of recurrence equations, one can inline the structured program to obtain a single flat Alpha program and then schedule it. But this method has two drawbacks: it looses the information on the structure of the program, and it leads to huge linear programs which become difficult to solve, as sean in section 3.

The other approach is to look for a *structured scheduling* as introduced in definition 2. This approach corresponds to a divide and conquer strategy for scheduling and has many advantages:

– The structure of the specifications, often meaningful from the point of view of the design flow, is kept unchanged.

[§] Je pense qu'on devrait mettre cela dans l'introduction et la motivation

- If some components of the system have already been designed, on can try to use their schedule and integrate them as such in the global design. For example, we may assume that a component for the `dot` subsystem has been designed, and that we want to obtain a matrix multiplication by re-using this component.
- The scheduling process is faster, as the problems to be solved are smaller.
- Finally, the schedules obtained are often easier to use in a design process. For instance, we can easily obtain a pipe-line between successive uses of the same system as shown in [11].

On the other hand, a structured linear scheduling may not exist and, even if it exists, it may not be optimal.

In this section we study the following problem: given an Alpha structured program, how can we find a structured linear scheduling for this program using current technique for finding out linear scheduling for recurrence equations?

A necessary condition for this scheduling to exist is obviously that the inlined system has a linear schedule, but as we shall see¶ this condition is not sufficient. We propose an algorithm for finding out structured linear scheduling using a linear scheduler such as the one presented in section 3 [15]. We show that the conditions that we provide are necessary and sufficient to find a structured linear scheduling, in the case of a bottom-up approach, i.e. when the called subsystems are scheduled before the calling system.

We consider two cases: when the SDG does not have cycles with use statement, and then the general case.

4.1 The SDG has no cycle with a use statement

The simplest case where we can find a linear structured scheduling is when the structured dependence graph does not have use statements in dependence cycles. This is the case in the program of Figure 3 for example, whose dependence graph is represented on Figure 4. Such a specification corresponds to n^2 independent use of the system `dot`.

A natural method for obtaining structured scheduling for this kind of structured programs is to schedule the `dot` system first, and then to deduce constraints for the scheduling of the system `MM`. As seen in section **??**, ‖a valid schedule of system `dot` is:

$$T^{\mathtt{dot}}_{\mathtt{V1}}(k) = 0 \quad T^{\mathtt{dot}}_{\mathtt{V2}}(k) = 0 \quad T^{\mathtt{dot}}_{\mathtt{Acc}}(k) = k \quad T^{\mathtt{dot}}_{\mathtt{res}} = 1 + N \quad . \tag{6}$$

There are N^2 such calls to `dot`, each one being attached to a unique pair (i, j) in the extension domain $\{i, j \mid 1 \leq i, j \leq N\}$. For a given `dot` instance, say instance (i_1, j_1), scheduling this instance at time $c(i_1, j_1)$ means exactly that the inputs `Mdup1[`k`,`i_1`,`j_1`]` and `Mdup2[`k`,`i_1`,`j_1`]`, $1 \leq k \leq N$ are available at time

¶ Où?

‖ Remettre cet exemple plus haut

$c(i_1, j_1)$ and that, according to (6), the output `Mres[`i_1`,`j_1`]` is available at time $c(i_1, j_1) + N + 1$.

For instance schedule

$$\begin{array}{ll} T^{\mathtt{MM}}_{\mathtt{Mdup1}}(k,i,j) = i+j \\ T^{\mathtt{MM}}_{\mathtt{Mdup2}}(k,i,j) = i+j \\ T^{\mathtt{MM}}_{\mathtt{Mres}}(i,j) \quad = i+j+N+1 \end{array} \tag{7}$$

is a valid structured scheduling for system `MM` as, associated to schedule (6) of `dot`, it meets the conditions of definition 2.

Based on the ideas of the above example, we can propose a method for finding structured scheduling for a system `sys` whose dependence cycles do not contain use statements:

- First decompose the SDG into strongly connected components, each one of which containing only one system call to `sub` which has the form of (4).
- Then solve the scheduling problem for each component, in the topological order of the components of the graph.

We give below the constraints that a linear structured scheduling of `sys` must satisfy.

4.2 Getting the dependencies from the SDG

Given a dependence in the SDG, if it is not labeled with a system name, the constraints are the usual scheduling constraints (dependency constraints, positivity constraints) [15].

Consider now the case when the dependency is labeled by a system name. Because of the restricted information contained in the SDG, this dependency is expressed by the following constraints: each output of the call $V_j(I, J)$ depends on each input $U_i(I', J)$ for $J \in \mathcal{E}$ (extension domain in (4)), $I \in \mathrm{Dom}(v_j)$ and $I' \in \mathrm{Dom}(u_i)$, where u_i and v_j are the formal arguments of system `sub` corresponding to U_i and V_j.

Theorem 1. *Let sys be an Alpha program, with a call to subsystem* `sub`*. Let T^{sub} be a schedule of* `sub`*. Then the following constraints are necessary and sufficient to obtain a valid structured schedule T^{sys}:*

$$\begin{array}{ll} \forall I \in \mathrm{Dom}(u_i), & T^{\mathit{sys}}_{U_i}(I, 0) = T^{\mathit{sub}}_{u_i}(I) \\ \forall I \in \mathrm{Dom}(v_i), & T^{\mathit{sys}}_{V_j}(I, 0) = T^{\mathit{sub}}_{v_i}(I) \\ \forall J \in \mathcal{E}, & T^{\mathit{sys}}_{U_i}(0, J) - T^{\mathit{sys}}_{V_j}(0, J) = T^{\mathit{sub}}_{u_i}(0) - T^{\mathit{sub}}_{v_i}(0) \end{array} \tag{8}$$

The proof of this theorem is given in appendix A.

To illustrate this result, consider again the example of Fig. 3. The constraints deduced from schedule (6) are:

$$\begin{array}{ll} \forall k, 1 \le k \le N, & T^{\mathtt{MM}}_{\mathtt{Mdup1}}(k,0,0) = 0, T^{\mathtt{MM}}_{\mathtt{Mdup2}}(k,0,0) = 0 \\ T^{\mathtt{MM}}_{\mathtt{Mres}}(0,0) = N+1, & \\ \multicolumn{2}{l}{\forall i,j, 1 \le i,j \le N, T^{\mathtt{MM}}_{\mathtt{Mdup1}}(0,i,j) = T^{\mathtt{MM}}_{\mathtt{Mdup2}}(0,i,j) = T^{\mathtt{MM}}_{\mathtt{Mres}}(i,j) - N - 1} \end{array}$$

The scheduling of (7) meets these constraints. Another valid structured scheduling is:

$$\begin{array}{ll} T^{\mathtt{MM}}_{\mathtt{Mdup1}}(k,i,j) &= 0 \\ T^{\mathtt{MM}}_{\mathtt{Mdup2}}(k,i,j) &= 0 \\ T^{\mathtt{MM}}_{\mathtt{Mres}}(i,j) &= N+1 \end{array} \tag{9}$$

4.3 Top down scheduling

[**]. If the SDG do not contains any cycle, the method presented here can be applied even if the schedule of the system used `sub` has not already been computed. In that case, when the computation dates of the actual inputs U_i are known, one just compute the schedule of `sub` with additional constraints on formal inputs u_i inherited from the computation dates of the actual inputs U_i. This top-down approach always provides a linear structured scheduling whenever then exists one and moreover, it choses the fastest linear schedule if we try to minimize the execution time. Indeed, if there is no use statement in a cycle, each strongly connected component can be scheduled in minimum time and the total time is the sum of the execution times of the components.

4.4 Structured linear scheduling in the general case

When the structured dependence graph contains cycles with use statements, imposing a schedule for the called system may prevent from finding a linear structured scheduling when one exists. Indeed, constraints of system (8) refer to an a priori given schedule for `sub`, and it may happen that this given schedule is not compatible with a structured schedule, although other solutions exist. To overcome this problem, we should be able to compute a schedule of `sys` parameterized by the schedule of `sub`. [††] We propose to use a bottom-up approach for the structured scheduling:

- First compute the schedule of all subsystems called in `sys`.
- Then schedule T^{sys} with the additional constraints of (8).

Of course, this bottom up approach will fail in some cases, when a very specific schedule of the subsystem `sub` is needed, but in practice, as a problem is decomposed into sub-problems already solved, a system may be decomposed in sub-systems already scheduled. [‡‡]

In fact, the class of program with cycles containing use statements that admits linear structured schedules is very restricted. The problem comes from dependencies from an output of a system call to an input of another one. To illustrate this, consider a dependency from an output of `sub` to an input of `sub`:

$$U_i[I,J] \leftarrow V_j[f(I,J), g(I,J)] \tag{10}$$

[**] Demander si c'est pas de cycles, ou pas de cycles avec des appels

[††] On pourrait essayer de résumer les contraintes données par `sub` en projettant les contraintes sur les paramètres d'entrée et de sortie?

[‡‡] Je ne comprends pas bien cela...

where J denotes the vector of extension indices. If $f(I, J)$ really depends upon J, then the time-shift between different inputs to `sub` may vary with the instance executed, as it may depend on J. It can also be proven that if $g(I, J)$ really depend upon I, then the time-shift between an input and an output of `sub` also depends on the instance J. In both cases, it may be impossible to use the same schedule for all instances of `sub`.

There is a situation when this problem disappears. Consider dependencies of the form

$$U_i[I, J] \leftarrow V_j[f(I), g(J)] \quad . \tag{11}$$

where $f(I)$ depends only on I and $g(J)$ on J. If all dependencies from outputs to inputs of called subsystems have this form, we say that the system has the *separated dependency property.* The following theorem gives a solution to the problem.

Theorem 2. *Consider a structured system* `sys` *which has the separated dependency property. Then* `sys` *admits a linear structured schedule only if, for each subsystem call to* `sub` *contained in a cycle of the* SDG*, the time delay on each dependence path from an input to an output of subsystem* `sub` *is constant.*

The proof of this theorem is given in appendix B. This condition is very restrictive, but this is not surprising because, using repeatedly a complex procedure, we would get a non linear time.

We can now propose algorithm A for computing a linear structured scheduling of an Alpha system `sys`.

Algorithm A

(Given: a system* `sys`*, and a call to subsystem* `sub` **)*
Perform a topological sort of the structured dependence graph of `sys`
`for` *each component c (in topological order)* `do`
    determine the constraints of the input to the component
    `for each` *call $\mathcal{C}$ to a subsystem* `sub`*) in c (written as (4))* `do`
        build the set $\mathcal{IO}_\mathcal{C}$ of pairs (U_i, V_j) such
            that there is a dependence from V_j to U_i in `sys` *as in (10)*
        schedule `sub` *by Algorithm A, with constraints from input to the*
            component and minimization of the latency between consumption
            of U_i and the production of V_j for $(U_i, V_j) \in \mathcal{IO}_\mathcal{C}$
        `If` *no schedule is found* `then return` *fail*
    `endfor`
    schedule c using the schedule found for `sub` *in constraints (8)*
    `If` *no schedule is found* `then return` *fail*
`endfor`

Where?
Je ne comprends pas

This algorithm is valid, because we have proven that constraints (8) are sufficient to provide a structured linear schedule.

Algorithm A can easily be modified in order to use the same schedule for different calls to the same system `sub`. To do so, we replace the second schedule of `sub` in algorithm A by the first schedule found for `sub`, thus giving priority to the first call in the SDG.

As already said, algorithm A may fail, for instance, if the schedule of subsystems `sub` is not properly chosen. In the next section, we look for non-linear structured scheduling.

5 Structured multi-dimensional scheduling

In the context of loop parallelization, non linear scheduling has been studied through *multi-dimensional* scheduling. In this section, we present methods to obtain structured multi-dimensional schedules, after introducing background material necessary to formalize this notion.

5.1 Multi-dimensional scheduling

The principles of multi-dimensional scheduling were settled down by Karp, Miller and Winograd [17] and explored further in [14, 8]. The idea is as follows. Consider an Alpha system which does not have a linear schedule. This means that no linear expression on the indices provides a partial order that respects the dependencies of the system. Instead of looking for a unique linear expression T_V associated with each variable, we look for a vector of linear expressions upon the indices which expresses the computation date. Let $\ll$ denote the lexicographic order between vectors, i.e.:

$$X \ll Y \text{ if there exists } i \text{ such that} \begin{cases} \mathrm{x}_k = y_k, \forall k < i \\ \mathrm{x}_k < y_k, \forall k \geq i \end{cases}$$

Then, if U depends on V in the SARE, we impose that $T_U(z) \ll T_V(z)$ $\forall z \in \mathrm{Dom}(U)$.

For instance, the following multi-dimensional schedule is valid for the program of Fig. 3:

$$T_1(i,j,k) = T_2(i,j,k) = \begin{pmatrix} i \\ k \end{pmatrix}$$
$$T_{\mathtt{Mres}}(i,j) = \begin{pmatrix} i \\ N+1 \end{pmatrix}$$

This schedule has two dimensions. It does not really give an absolute date of computation for each variable but merely a relative order: first compute the coefficients of the first row of `Mres` ($i = 1$), then when this is finished, compute the coefficients of the second row ($i = 2$, because $T_{\mathtt{Mres}}[1,j] \ll T_1[2,j,k]$, $\forall j,k$), and so on. An intuitive explanation of what a multi-dimensional schedule represents, one can interpret the components of the time vector as hours, minutes, seconds, etc. However this interpretation is realistic only if all the variables have the same rectangular domains, which is not always the case.

5.2 Computing multi-dimensional schedules

Computing multi-dimensional schedule is based on the following idea. If a dependency, say for instance $T_{\texttt{Mres}}(i,j) - T_1(i,j,k) \geq 1\ \forall i,j,k)$, cannot be met using a linear schedule, we introduce an additional dimension in the timing vector. T becomes then $\begin{pmatrix} T^1 \\ T^2 \end{pmatrix}$). Then we try to satisfy the relaxed constraint $T^1_{\texttt{Mres}}(i,j) - T^1_1(i,j,k) \geq 0,\ \forall i,j,k$ by means of the first component of T and leave it up to T^2 to *strictly satisfy* the dependency $T^2_{\texttt{Mres}}(i,j) - T^2_1(i,j,k) > 0$, $\forall i,j,k$. We say that this dependency is *satisfied at level 2*.

The schedule shown in (12) respects these constraints on T^1 and T^2. Hence, computing multi-dimensional schedules can be done with a linear scheduler provided that we implement the process described here. Feautrier [14] does this by minimizing the number of dimensions of the resulting schedule.

Multi-dimensional scheduling provides a lot of flexibility to the loop parallelization. However, the space of possible valid multi-dimensional schedules is huge and few strategies for multi-dimensional scheduling have yet been proposed, which limits the practical use of such methods.

In the next section we propose a strategy for finding structured multidimensional scheduling.

5.3 Multi-dimensional schedule and structuring

In section 4, we looked for linear schedules by imposing constraints on the computation dates of inputs and outputs of system calls. In the multi-dimensional case, we do not know in advance which dependency is satisfied at which level, hence, it is impossible to enforce values for timing vectors, and we must propose another method for finding structured scheduling.

To obtain structured multi-dimensional schedules, one can consider that each call to a subsystem takes no time, with respect to a given vector of time. In other words, the inputs and outputs are assumed to be available at the same instant of time. Then, we add extra dimensions to the time vector in order to satisfy the dependency constraints inside the subsystem. This method can always be applied since the graph of the system calls of an Alpha program is acyclic (two systems cannot call each others.)

Let us formalize this technique. Assume that we have a procedure (Algorithm A for example) that computes a linear structured scheduling. Given an Alpha system, algorithm B tries to find out a structured multidimensional scheduling (which has as many as possible linear structured schedules for system calls).

Je ne comprends pas cette phrase

Algorithm B

1. *Perform a topological sort of the* SDG *of* `sys`
2. `for each` *component c of the* SDG `do`
   determine the constraints on the inputs of the component
   compute a schedule T_c for this component by Algorithm A
   `if` *no schedule is found* `then do`
   `for each` *system call to s_i of the component* `do`
   schedule s_i with Algorithm B and get a k_i-dimensional schedule T_c^i
   `end for`
   get a k-dimensional schedule T'_c of the component c
   with the constraints that inputs and outputs of a system call
   have exactly the same schedule
   Build a $k + \max_i(k_i)$-dimensional schedule $T_c = \begin{pmatrix} T'_c \\ T_c^i \end{pmatrix}$ (complete
   possibly T_c^i with 0's)
   `end if`
   `enddo`
3. *If the T_c schedules have different dimensions for different c, padd the smaller ones with 0's.*

5.4 Validity of Algorithm B

In this section, we prove that the schedules obtained by algorithm B are valid structured scheduling, i.e. that all dependencies are respected. We have already proven that algorithm A provides linear schedules which respect dependencies. Remains to prove that the schedule found by our procedure (if algorithm A fails) respects dependencies inside a strongly connected component and that dependencies between strongly connected components are ensured.

Inside a strongly connected component, dependencies in a subsystem `sub` (from inputs to outputs) are respected because all variables have the same coefficients in T'_c, and the recursive use of algorithm B provides coefficients of T_c^i that meet dependencies (a recurrence hypothesis should be set to prove that correctly, we assume that this property is clear enough not to prove it so formally). Inside a strongly connected component `c`, a dependency occurring in the `sys` system is ensured by T'_c. As all our dependence constraints are *strict* (i.e. we forbid computation which last 0, except for the calls), appending T_c^i (and possibly 0's) at the end, will not change the lexicographic order.

Dependency between strongly connected components are ensured because, before scheduling one component, we compute the constraints (on the first level of schedule) derived from the schedules of all previous (in the topological order sense) components. Hence, the addition of 0's performed in step (3) concern variable for which dependency constraints have been (strictly) satisfied at a higher level.

?

This algorithm provides one possible strategy for structured multi-dimensional scheduling. We will see in section 6 that it gives good results on a particular example. Of course, there are drawbacks with this approach. First, as for the linear case depicted in section 4, this will not work on all Alpha program which admit multi-dimensional schedule. Again, this is due to the shortcut in the SDG for dependencies inside a system call: some outputs may not depend on some inputs, hence a cycle in the SDG may not be a real cycle. For instance, if the dependency occurring from an output to an input is identity, this method will not work. Currently we do not have a solution to this problem. This is inherent to the definition we gave for a SDG, But, again, we point out that writing a call where inputs depend on outputs is not coherent with the usual notion of structuring. This will never happen if, for instance, the SAREs come from imperative code which has been automatically translated.

Another problem with this approach is that it forbids the possible pipeline between successive instances of call of the same (or different) system. The output of a system will not be used before the system is completely executed, this is a great drawback in VLSI synthesis since pipeline is very often used. In our approach, pipeline can only be achieved with linear schedules obtained by the call to algorithm A.

6 Structured scheduling for the singular value decomposition

In this section we show how to derive a structured scheduling for a realistic example with the method presented in this paper. The example chosen is a part of an algorithm for computing the singular value decomposition (SVD) of a $M \times 2N$ matrix. The SVD is used in many area (mobile telephone for instance [10]). Several method were proposed to find the SVD of a matrix (see [16] for a survey). One of these methods (explained in [2]) is a Jacobi-like algorithm with iterative applications of successive orthogonalizations of pairs of columns in the matrix.

This method has been specified in Alpha, in Program 5, the `SWEEP` system represented one iteration of the Jacobi algorithm. The SDG of `SWEEP` is represented on Figure 7 (some information have not been represented for the sake of clarity). Note that the call to `STEP` in `SWEEP` has the separated dependence property. `STEP` and the other systems called are represented in Program 6, the system `STEP` corresponds to the orthogonalization of N particular pairs of columns (each column is involved one time) as A is of order $M \times 2N$. This specification is inspired from Blas libraries [20]. From this specification, it seems clear that the `STEP` system should have a linear schedule as it consists of `N` independent orthogonalizations (each having an $O(M)$ complexity).

In this section, we adopt the convention that one equation takes one clock cycle, this can be modified and greatly influence the resulting scheduling. Following algorithm B, we start by the scheduling of system `SWEEP` and we suppose, for instance, that coefficient $A_{i,j}$ of the input matrix `A` is available at step i (constraint from a previous computation): $T_A(i, j) = i$. The SDG of `SWEEP` (see

Figure 7) contains only one strongly connected component which includes all local variables: `C1,C2,C1r,C2r`. The first thing to compute are the constraints on `C1,C2` deriving from the input `A`. Here, we must have: $T_{C1}^{SWEEP}(i,j,k) \geq i+1$ and $T_{C2}^{SWEEP}(i,j,k) \geq i+1$. Now, we go down to the schedule of `STEP` with the constraints on formal inputs `C1,C2`; $T_{C1}^{STEP}(i,j) \geq i+1$ and $T_{C2}^{STEP}(i,j) \geq i+1$

Here, we can notice that the SDG of `STEP` (see Figure 8) is acyclic, as are all the SDG of the systems called by `STEP`. Hence, we will be able to find a linear scheduling using the Algorithm A of section 4 (provided that each system used in `STEP` has a linear scheduling). We transmit to the schedule of `DROT` the constraints on formal input `v1,v2`: $T_{v1}^{DROT}(i) \geq i+1$ and $T_{v2}^{DROT}(i) \geq i+1$. `DROT` uses the two subsystems `DDOT` and `DNRM2`. The schedule of `DNRM2` with the constraint of formal input `v`: $T_{v}^{DNRM2}(i) >= i+1$ gives:

$$T_{v}^{DNRM2}(i) = 1+i \quad T_{vcumul}^{DNRM2}(i) = 2+i \quad T_{norm}^{DNRM2} = 3+M \quad .$$

The schedule of `DDOT` is similar:

$$T_{v1}^{DDOT}(i) = 1+i \quad T_{v2}^{DDOT}(i) = 1+i \quad T_{vcumul}^{DDOT}(i) = 2+i \quad T_{pdtscal}^{DDOT} = 3+M \quad .$$

Then we try to schedule system `DROT` by algorithm A, i.e. with additional constraints on formal inputs `v1,v2` ($T_{v1}^{DROT}(i) = i+1$ and $T_{v2}^{DROT}(i) = i+1$) and local variables `alpha,beta,gama` ($T_{alpha}^{DROT} - T_{v1}^{DROT}(0) = 2+M$, $T_{beta}^{DROT} - T_{v1}^{DROT}(0) = 2+M$, $T_{gama}^{DROT} - T_{v1}^{DROT}(0) = 2+M$), the result is:

$$\begin{array}{llll}
T_{v1}^{DROT}(i) = 1+i & T_{v2}^{DROT}(i) = 1+i & & \\
T_{alpha}^{DROT} = 3+M & T_{beta}^{DROT} = 3+M & T_{gama}^{DROT} = 3+M & \\
T_{eps}^{DROT} = 4+M & T_{t}^{DROT} = 5+M & T_{c}^{DROT} = 6+M & T_{s}^{DROT} = 7+M \\
T_{C1}^{DROT}(i) = 8+M & T_{C2}^{DROT}(i) = 8+M & &
\end{array} \tag{12}$$

Now we are able to schedule `STEP`. Here, the `use` has an extension domain, hence we will add to the constraints deduced from (12), the extension domain constraints:

$$\begin{array}{l}
T_{C1}^{STEP}(0,j) - T_{C2}^{STEP}(0,j) = 0, \\
T_{C1}^{STEP}(0,j) - T_{C1r}^{STEP}(0,j) = 7+M, \quad \forall j,\ 1 \leq j \leq N \\
T_{C1}^{STEP}(0,j) - T_{C2r}^{STEP}(0,j) = 7+M
\end{array}$$

Schedule of `STEP` can be obtained by algorithm A (as we have suggested earlier), the result is:

$$T_{C1}^{STEP}(i,j) = T_{C2}^{STEP}(i,j) = 1+i \quad T_{C1r}^{STEP}(i,j) = 8+M \quad T_{C2r}^{STEP}(i,j) = 8+M \quad . \tag{13}$$

We finally come to the schedule of the `SWEEP` system. Attempting to find a linear schedule for this system will fail as the execution time of one instance of `STEP` is not constant $(N+8)$, thus `SWEEP` does not have a linear schedule. Following algorithm B, we will add a dimension for the call to `STEP` and schedule `SWEEP` imposing that, for each instance `k` of `STEP`, inputs and outputs are computed simultaneously. Hence the constraints:

$$T_{C1}^{SWEEP}(i,j,k) = T_{C2}^{SWEEP}(i,j,k) = T_{C1r}^{SWEEP}(i,j,k) = T_{C2r}^{SWEEP}(i,j,k)$$
$$\forall (i,j,k)\ 1 \le i \le M,\ 1 \le j \le N,\ 1 \le k \le 2N-1$$

Note that we can forget the input constraints coming from `A` because these have already been included in the schedule of `C1,C2` and do not influence directly other variables. The resulting linear schedule is:

$$\begin{array}{l} T_{A}^{SWEEP}(i,j) = 0 \quad T_{C1}^{SWEEP}(i,j,k) = k\ T_{C2}^{SWEEP}(i,j,k) = k \\ T_{C1r}^{SWEEP}(i,j,k) = k\ T_{C2r}^{SWEEP}(i,j,k) = k\ T_{B}^{SWEEP}(i,j,l) = 2N \end{array} \tag{14}$$

Schedule (14) and (13) finally give as multi-dimensional scheduling for system `SWEEP`: $T = \begin{pmatrix} T^{SWEEP} \\ T^{STEP} \end{pmatrix}$.

$$\begin{array}{ll} T_A(i,j) = \begin{pmatrix} 0 \\ i \end{pmatrix} & T_B(i,j,l) = \begin{pmatrix} 2N \\ 0 \end{pmatrix} \\ T_{C1}(i,j,k) = \begin{pmatrix} k \\ i+1 \end{pmatrix} & T_{C2}(i,j,k) = \begin{pmatrix} k \\ i+1 \end{pmatrix} \\ T_{C1r}(i,j,k) = \begin{pmatrix} k \\ M+8 \end{pmatrix} & T_{C2r}(i,j,k) = \begin{pmatrix} k \\ M+8 \end{pmatrix} \end{array} \tag{15}$$

One can check that this schedule is valid. It respects each dependence, for instance, $C1[i,j,k] \leftarrow C1r[i,j-1,k-1]$ is respected because:

$$\begin{pmatrix} k-1 \\ M+8 \end{pmatrix} \ll \begin{pmatrix} k \\ i+1 \end{pmatrix} \ \forall (i,k) \quad .$$

Also, the input constraints (from `A`) is respected because $A[i,j]$ is scheduled at $\begin{pmatrix} 0 \\ i \end{pmatrix}$. Hence we obtained a valid structured multi-dimensional schedule for system `SWEEP`.

Another method for scheduling the computations of `SWEEP` would be to look for a non-structured scheduling, hence to inline all the structuring of `SWEEP` in order to obtain a single big system of recurrence equation and to schedule it with a classical multi-dimensional scheduling tool. The first problem with this approach is the computation time. In the present case, the LP generated from the unrolled `SWEEP` system is composed of 1081 variables and 1138 constraints, while the maximum size of the linear programming problem generated with the previous method was for `SWEEP`: 196 variables and 226 constraints. Even if the structured scheduling method contains many manipulations, while this one consists in a single schedule, this complexity problem will become critical for bigger programs. The other problem is also crucial: as we cannot know in advance how many dimensions will compose the schedule, we must impose the constraints deriving from input `A` on the highest level: $T_A(i,j) \ge i$, which will mean after scheduling: $T_A(i,j) \gg \begin{pmatrix} i \\ 0 \end{pmatrix}$, while the effective constraints in the first method was: $T_A(i,j) \gg \begin{pmatrix} 0 \\ i \end{pmatrix}$.

The resulting flat schedule is also bi-dimensional, it is shown in (16), were we only print the schedule relative to the original variables of `SWEEP`.

$$
\begin{array}{ll}
T_A(i,j) = \begin{pmatrix} -1+i \\ 0 \end{pmatrix} & T_B(i,j) = \begin{pmatrix} -8+M+16N \\ 0 \end{pmatrix} \\
T_{C1}(i,j) = \begin{pmatrix} 8k+M \\ 0 \end{pmatrix} & T_{C2}(i,j) = \begin{pmatrix} 8k+M \\ 0 \end{pmatrix} \\
T_{C1r}(i,j) = \begin{pmatrix} -1+8k+M \\ 0 \end{pmatrix} & T_{C2r}(i,j) = \begin{pmatrix} -1+8k+M \\ 0 \end{pmatrix}
\end{array}
\tag{16}
$$

The second dimension is used for scheduling intermediate variables that come from the inlining of the systems used. Just note that the number of *minutes* in one *hour* (i.e the number of time steps enumerated by the second dimension of the schedule) is roughly the same for schedule (15) (M+8) and schedule (16) (M) . The main differences between the two schedules come from the N parameter which appears in the final schedule of `B` (in (16)) and the $8k$ in the local variables (instead of k in (15)). The $8k$ coefficient comes from the fact that many dependencies are by default satisfied at the first level (while they were satisfied at lower level in the schedule (15)), the N come from the input constraint from `A` which is also ensured at the first level because of the reason mentioned above.

The flat schedule (16) increases the number of step of the scheduled `SWEEP` from $(2N+1)*(M+8) = O(MN)$ to $(16M+N-15)*M = O(M^2+NM)$. In our example, structured scheduling combines the advantages of preserving the structure of the program and providing a faster schedule.

7 Conclusion

We have presented the basic concepts of structured scheduling for recurrence equations. We have first given precise definitions for the structured dependence graph of a structured SARE, and for the notion of structured scheduling. With respect to these definitions, we have provided an algorithm for finding a valid (linear or multi-dimensional) structured scheduling. The great advantage of our approach is that it permits to reuse a scheduling tool for unstructured SAREs and thus can be implemented very quickly.

We also proved that, if we are looking for linear structured scheduling with a "bottom-up" approach, our algorithm was *optimal* in the sense that if ever there exists a linear structured scheduling, we will find it. It has been exemplified on a realistic example: the SVD extracted form the Blas library.

Structured scheduling allows to keep the structuring information and to simplify the scheduling computation complexity. It also provides more readable and natural scheduling. Moreover, at least for the example we tackled, it may give better results (in term of resulting execution time) than a naive flat scheduling.

This work has been done in order to provide a strategy for scheduling large Alpha systems in the MMAlpha environment. Many open questions remain. First, we do not know if it is possible to statically determine all the cases where a structured SARE admits a structured linear scheduling (neither for structured multi-dimensional scheduling of course). Algorithm B proposed for multi-dimensional structured scheduling is one possible method among many variants, it remains to evaluate this method and to compare it to other approaches. In particular, in this paper, we wanted to use a classical schedule tool as the basis of a structured scheduling tool. If we get rid of this implementation constraint, it may be possible to define completely different scheduling algorithms. Finally, we are certainly aware of the fact that the implementation of a scheduled structured SARE in hardware has still to be studied in depth, especially concerning the *real cost* of a multi-dimensional time.

References

1. E. Ashcroft and W. Wadge. Lucid, a formal system for writing and proving programs. *SIAM j. Comp.*, 3:336–354, 1976.
2. R. P. Brent and F. T. Luk. The solution of the singular value and symmetric eigenvalue problems on multiprocessors arrays. *SIAM J. Sci. Statist. Comput*, 6:69–84, 1985.
3. P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th Symposium on Principles of Programming Languages*. ACM, Munich, Janvier 1987.
4. M. Chen, Y. Choo, and J. Li. *Crystal: Theory and Pragmatics of Generating Efficient Parallel Code*, page Chapter 7. ACM Press, 1991.
5. A. Darte. *Techniques de parallélisation automatique de nids de boucles*. Thèse de doctorat, LIP ENS-Lyon, 1993.
6. A. Darte, L. Khachiyan, and Y. Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.
7. A. Darte and Y. Robert. Scheduling uniform loop nests. Technical Report 92-10, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, Feb. 1992.
8. A. Darte and F. Vivien. Revisiting the decomposition of Karp, Miller and Winograd. In *Application Specific Array Processor*, pages 13–25, 1997.
9. F. D. De Dinechin, P. Quinton, and T. Risset. Structuration of the Alpha language. In W. Giloi, S. Jahnichen, and B. Shriver, editors, *Massively Parallel Programming Models*, pages 18–24. IEEE Computer Society Press, 1995.
10. E. F. Deprettere, editor. *SVD and Signal Processing Algorithms, Applications and Architectures*, North-Holland, 1988. Elsevier Science Publishers B.V.
11. F. Dupont De Dinechin. Libraries of schedule-free operators in Alpha. In *Application Specific Array Processor*, 1997.
12. F. Dupont De Dinechin and P. Le Moenner. Automatic synthesis of regular architectures optimized at the bit level. In *Workshop on Design Methodologies for Signal Processing*, Zakopane, Poland, Aug. 1996.
13. F. Dupont De Dinechin, S. Robert, and T. Risset. Structured scheduling of recurrence equations. Technical Report 1140, Irisa, Rennes, France, 1997.
14. P. Feautrier. Some efficient solution to the affine scheduling problem, part II, multidimensional time. *Int. J. of Parallel Programming*, 21(6), Dec. 1992.
15. P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, Oct. 1992.
16. G. Golub and C. V. Loan. *Matrix computations*. The Johns Hopkins University Press, 1989.

17. R. Karp, R. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, July 1967.
18. H. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.
19. L. Lamport. The Parallel Execution of DO Loops. *Communications of The ACM*, 17(2):83–93, Feb. 1974.
20. C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft*, 5:308–323, 1979.
21. P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. SIGNAL: A data flow oriented language for signal processing. In *IEEE Workshop on VLSI 1984*, 1984.
22. P. Lee and Z. Kedem. Mapping nested loop algorithms into multidimensional systolic arrays. *IEEE Transaction On Parallel and Distributed System*, 1(1):64–76, Jan. 90.
23. C. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones.* Thèse de doctorat, Ifsic, Université de Rennes 1, Dec. 1989.
24. C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In S. Kung, J. E.E. Swartzlander, J. Fortes, and K. Przytula, editors, *Application Specific Array Processors*, pages 100–110. IEEE Computer Society Press, Sept. 1990.
25. D. Moldovan. On the analysis and synthesis of VLSI systolic arrays. *IEEE Transactions on Computers*, 31:1121–1126, 1982.
26. G. Perrin, S. Genaud, and E. Violard. PEI: a theoretical framework for data-parallel programming. Technical report, ICPS, Strasbourg, 1994.
27. Y. Saouter. *A propos de systèmes d'équations récurrentes.* Thèse de doctorat, Ifsic, Université de Rennes 1, Oct. 1992.
28. M. Wolf and M. Lam. Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, Oct 1991.

A constraints for structured linear scheduling

In the following, we will call *degenerated* a domain which contains equalities in its definition (e.g. $\mathcal{D} = \{i, j \mid i = j; i \geq 0\}$ is degenerated). We have to prove two lemmas, the first one is the usual case where there are no degenerated domains concerned with the call, the other one is a generalization to any situations.

Lemma 1. *Consider the call (17) to* `sub` *in* `sys` *where domains of input/output variables and the extension domain are not degenerated:*

$$\texttt{use}\ \{J|AJ \leq b\}\ \texttt{sub[f(J,N)]}\ (U_1, \dots, U_n)\ \texttt{return}\ (V_1, \dots, V_m) \quad . \tag{17}$$

If the schedule of `sub`*: T^{sub} is already computed. The constraints (18) added to the usual dependency constraints between variables of* `sys` *suffice to obtain a valid structured scheduling T^{sys} of* `sys`*. Moreover these constraints are respected by every structured scheduling (i.e., they are necessary).*

$$\begin{array}{ll} T^{sys}_{U_i}(I,0) = T^{sub}_{u_i}(I) & \forall I \in Dom(u_i) \\ T^{sys}_{V_j}(I,0) = T^{sub}_{v_i}(I) & \forall I \in Dom(v_i) \\ T^{sys}_{U_i}(0,J) - T^{sys}_{V_j}(0,J) = T^{sub}_{u_i}(0) - T^{sub}_{v_i}(0) & \forall J \in \{J|AJ \leq b\} \end{array} \tag{18}$$

Where u_i and v_j are the formal arguments of system `sub` *corresponding to U_i and V_j*

Proof: we look for structured scheduling (definition 2), hence T^{sys} must meet the following conditions:

$$\begin{array}{l} \text{for all actual input } U_i[I,J]\text{:} \\ \quad T^{sys}_{U_i}(I,J) = c(J) + T^{sub}_{u_i}(I)\ \forall (I,J) \in Dom_{U_i} \\ \text{for all actual output } V_i[I,J]\text{:} \\ \quad T^{sys}_{V_i}(I,J) = c(J) + T^{sub}_{v_i}(I)\ \forall (I,J) \in Dom_{V_i} \end{array}$$

These equalities occur for all $(I,J) \in Dom_{U_i}$ (resp. $(I,J) \in Dom_{V_i}$). Unless these domains are degenerated, these equalities can be extended to the whole space (any value of (I,J)). Hence, for non-degenerated domains, we can deduce (setting I then J to 0):

– $T^{sys}_{U_i}(0,J) - T^{sys}_{V_j}(0,J) = T^{sub}_{u_i}(0) - T^{sub}_{v_j}(0)$ and,
– $T^{sys}_{U_i}(I,0) = T^{sub}_{u_i}(I), \qquad T^{sys}_{V_j}(I,0) = T^{sub}_{v_i}(I) \quad .$

We obtain that the condition (18) are necessary when the domains are not degenerated.

We have now to show that, for each dependence d_1 between input and output of `sub`, constraints (18), impose that d_1 is respected by T^{sys}. Here, we will note $\mathcal{D}(N)$ to express the fact that domain $\mathcal{D}$ is parameterized by N.

Consider the following dependence constraints (non-direct dependency $d1$):

$$T^{sub}_{v_i}(I) - T^{sub}_{u_j}(d_1(I,M)) \geq d(I,M) \quad \forall I \in \mathcal{D}_{d1}(M), \tag{19}$$

In expression (18), the value for the parameter of `sub` is $f(J,N)$

where M represents the parameters of `sub`, d_1 is the value of the dependence and d is the *duration* (affine function of I and M). As T^{sub} is a valid schedule of `sub`, this constraint is respected by T^{sub}. From the definition of the structuring mechanism (see [9]) in Alpha, the corresponding constraint that must be respected by the schedule T^{sys} of `sys`, which contains the call (17), is:

$$T^{sys}_{V_i}(I,J) - T^{sys}_{U_j}(d_1(I,f(J,N),J) \geq d(I,f(J,N)) \quad \begin{array}{l} \forall J \in \{J|AJ \leq b\} \\ \forall I \in \mathcal{D}_{d1}(f(J,N)) \end{array},$$

where N represents the parameters of `sys` and $f(J,N)$ is the parameter assignment function in (17). As we have $D_{d_1}(f(J,N)) \subset D_{v_i}(f(J,N))$, from the constraints (18) we can deduce:

$$T^{sys}_{V_i}(I,J) - T^{sys}_{U_j}(d_1(I,f(J,N),J) = T^{sub}_{v_i}(I) - T^{sub}_{u_j}(d_1(I,f(J,N))) \quad \begin{array}{l} \forall J \in \{J|AJ \leq b\} \\ \forall I \in \mathcal{D}_{d1}(f(J,N)) \end{array}.$$

By the construction of the `use`, if J is in the extension domain: $\{J|AJ \leq b\}$ and if N is in the parameter domain of `sys`, $f(J,N)$ is in the parameter domain of `sub`. Hence we can use the inequality (19), and we get the result:

$$T^{sys}_{V_i}(I,J) - T^{sys}_{U_j}(d_1(I,f(J,N),J) \geq d(I,f(J,N)) \quad \begin{array}{l} \forall J \in \{J|AJ \leq b\} \\ \forall I \in \mathcal{D}_{d1}(f(J,N)) \end{array}.$$

Hence the constraints generated by dependence d_1 is respected by a scheduling of `sys` provided it respect respects constraints (18). $\square$

Lemma 2. *If we consider a call to* `sub` *in* `sys` *(see (17)) in which degenerated domains occurs. Then the constraints (18) are still sufficient, but may not be respected by a valid structured scheduling* T^{sys} *of* `sys`*. However, in that case, it is always possible to find another expression of the same schedule in which constraints (18) are respected.*

Proof: the second part of the previous proof can be repeated here, and we obtain that the dependency constraints are respected if constraints (18) are added to the schedule. The problem with degenerated domains come from the fact that we may not be able to extend the relation of definition (2) to $J = 0$.

Say we have this relation happening on the domain of U_i:

$$T^{sys}_{U_i}(I,J) = c(J) + T^{sub}_{u_i}(I) \quad \forall (I,J) \in Dom_{U_i} \quad .$$

This relation can be extended to the affine space containing Dom_{U_i} that we note $Aff(Dom_{U_i})$. By hypothesis this space contains equalities but one important property (due to the structuring mechanism) is that none of these equalities involve together extension indices (J) and local indices (I). Hence, this set of equalities can be split in two sets: n_J equalities occurring on extension indices and n_I equalities occurring on local indices. the real dimension of $Aff(Dom_{U_i})$ is therefore $n - n_J - n_I$ (where n is the dimension of the global space).

Let us note m_J the number of extension indices (length of J) and $\{eq_1, \ldots, eq_{n_J}\}$ the set of equalities occurring on J. without loss of generality, we can suppose that this set of equalities has been simplified and the J components permuted such that equality eq_i defines J_i in term of $\{J_{n_J+1}, \ldots, J_{m_J}\}$. Suppose, for instance, that we cannot extend the relation along the first component of J. it means that we have: $T^{sys}_{U_i}(I, j_1, \ldots, j_m) = c(j_1, \ldots, j_m) + T^{sub}_{u_i}(I)$ and $T^{sys}_{U_i}(I, 0, j_2, \ldots, j_{m_J}) \neq c(0, j_2, \ldots, j_{m_J}) + T^{sub}_{u_i}(I)$. Hence, it means that the coefficient of j_1 in c is not null. In that case, just replace j_1 in c by its definition in term of $\{J_{n_J+1}, \ldots, J_{m_J}\}$ as defined by eq_1, we will obtain a new expression of the timing function of U_i which take the same value as $T^{sys}_{U_i}$ on $Aff(Dom_{U_i})$ and which respects the equality wanted along one more dimension. Just by counting the number of dimension added, one can check that we will use at most n_j equality on J to obtain a timing function $\mathcal{T}^{sys}_{U_i}$ which is equal to $T^{sys}_{U_i}$ on $Aff(Dom_{U_i})$ and respects: $\mathcal{T}^{sys}_{U_i}(I, 0) = T^{sub}_{u_i}(I)\ \forall I \in Dom(u_i)$

The same manipulation can be done with the n_I equalities on I to obtain the other equality: $\mathcal{T}^{sys}_{U_i}(0, J) = c(J) + T^{sub}_{u_i}(0)\ \forall J \in \{J | AJ \leq b\}$ And we have the result. $\square$

B Separated dependency property

$$U_i[I, J] \leftarrow V_j[f(I), g(J)] \quad . \tag{20}$$

Given a call to a subsystem `sub` in `sys` (see 17), the fact that all dependencies from output of subsystems calls V_j to input U_i have the form of (20) is called the *separated dependency property.* Real application where separated dependency property occur where tackled in [12]. We prove here the following lemma:

Lemma 3. *If a system* `sys` *has the separated dependency property for all its calls, then it admits a linear structured scheduling only if, for each subsystem call (to* `sub`*) contained in a cycle of the* SDG*, the time delay on each dependence path from an input to an output of subsystem* `sub` *is constant.*

Proof: We have two dependence path, $p_1 :\ U_i[I, J] \leftarrow V_j[f(I), g(J)]$, $\forall (I, J) \in Dom_{p_1}$ and also $p_2 :\ V_j[I, J] \leftarrow U_i[I', J]$, $\forall (I, I'), \forall J \in \mathcal{E}$. Suppose that the condition is not respected, i.e. $T^{sub}_{V_j}(I) - T^{sub}_{U_i}(I') >= f(J, N)$ (N is the parameter of `sys`, J are the extension indices, f is not constant). we have the constraints:

$$\begin{aligned}
&T^{sys}_{U_i}[I, J] \geq T^{sys}_{V_j}[f(I), g(J)] \geq f(J, N) + T^{sys}_{U_i}[I', g(J)] \\
&\geq f(J, N) + T^{sys}_{V_j}[f(I'), g^2(J)] \geq \underbrace{\ldots}_{\text{P cycles unrolled}} \\
&\geq P * f(J, N) + T^{sys}_{V_j}[f(I^{(P)}), g^{P+1}(J)]
\end{aligned}$$

Hence, unless the cycle is not a real cycle (i.e. it can only be unrolled a constant number of time), the above constraint is not linear in the parameters (but quadratic), hence, there is no hope to find a linear schedule. $\square$

```
system SWEEP : {M,N | 4<=M; 4<=N}
               (A : {i,j | 1<=i<=M; 1<=j<=2N} of real)
       returns (B : {i,j,l | 1<=i<=M; 1<=j<=N; 0<=l<=1} of real);
var
  C1 : {i,j,k | 1<=i<=M; 1<=j<=N; 1<=k<=2N} of real;
  C2 : {i,j,k | 1<=i<=M; 1<=j<=N; 1<=k<=2N} of real;
  C1r : {i,j,k | 1<=i<=M; 1<=j<=N; 1<=k<=2N-1} of real;
  C2r : {i,j,k | 1<=i<=M; 1<=j<=N; 1<=k<=2N-1} of real;
let
  use {k | 1<=k<=2N-1} STEP[M,N] (C1, C2)
                        returns  (C1r, C2r) ;
  C1[i,j,k] =
      case
        {| k=1} : A[i,2j-1];
        {| 2<=k} :
           case
             {| j=1} : C1r[i,j,k-1];
             {| j=2} : C2r[i,j-1,k-1];
             {| 3<=j} : C1r[i,j-1,k-1];
           esac;
      esac;
  C2[i,j,k] =
      case
        {| k=1} : A[i,2j];
        {| 2<=k} :
           case
             {| j=N} : C1r[i,j,k-1];
             {| j<=N-1} : C2r[i,j+1,k-1];
           esac;
      esac;
  B[i,j,l] =
      case
        {| l=0} : C1[i,j,2N-1];
        {| l=1} : C2[i,j,2N-1];
      esac;
tel;
```

Fig. 5. System `SWEEP` performing one step of the (one-sided) Jacobi Algorithm for the SVD

```
system DNRM2 : {M|4<=M}
      (v:{i|1<=i<=M} of real)
       returns (norm:real);
var
  vcumul:{i|0<=i<=M} of real;
let
  vcumul[i]=
    case
      {|i=0} : 0[];
      {|1<=i} : vcumul[i-1] +
                 v[i] * v[i];
    esac;
  norm[]=vcumul[M];
tel;


system DDOT :  {M|4<=M}
      (v1:{i|1<=i<=M} of real;
       v2:{i|1<=i<=M} of real)
       returns (pdtscal:real);
var
  vcumul:{i|0<=i<=M} of real;
let
 vcumul[i] =
  case
   {| i=0} : 0[];
   {| 1<=i} : vcumul[i-1] +
              v1[i] * v2[i];
  esac;
 pdtscal[] = vcumul[M];
tel;
```

```
system DROT:{M|4<=M}
     (v1:{i|1<=i<=M} of real;
      v2:{i|1<=i<=M} of real)
    returns
     (C1:{i|1<=i<=M} of real;
      C2:{i|1<=i<=M} of real);
var
 alpha,beta,gama,eps,t,c,s :real;
let
 use DNRM2[M] (v1)returns(alpha);
 use DNRM2[M] (v2)returns(beta);
 use DDOT[M] (v1,v2)returns(gama);
 eps[]=0.5*(beta-apha)/gama;
 t[]=if (eps = 0) then 1 else
     if (eps > 0) then
     1/(eps+sqrt(1+eps*eps)) else
     -1/(-eps+sqrt(1+eps*eps));
  c[] = 1 / sqrt(1 + t * t);
  s[] = t * c;
  C1[i] = c[] * v1 - s[] * v2;
  C2[i] = s[] * v1 + c[] * v2;
 tel;

system STEP:{M,N | 4<=M; 4<=N}
(C1:{i,j|1<=i<=M; 1<=j<=N} of real;
 C2:{i,j|1<=i<=M; 1<=j<=N} of real)
returns
 (C1r:{i,j|1<=i<=M; 1<=j<=N} of real;
 C2r:{i,j|1<=i<=M; 1<=j<=N} of real);
let
 use {j | 1<=j<=N} DROT[M](C1, C2)
                 returns  (C1r, C2r);
tel;
```

Fig. 6. system DROT (using DDOT AND DNRM2) computing the orthogonalization of two column of a matrix, is used by N times by system `STEP` which scans all the matrix.

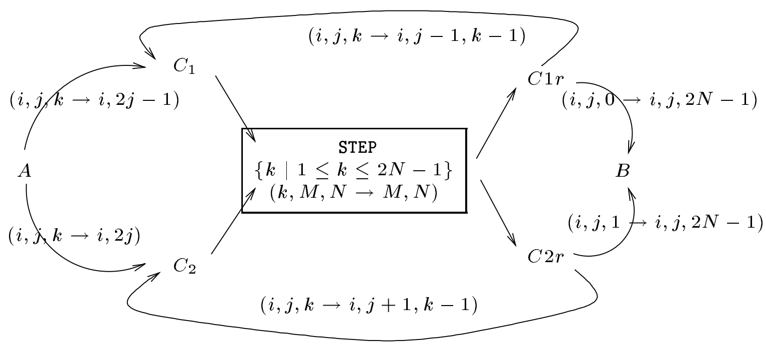

Fig. 7. Structured dependence graph of the `SWEEP` system

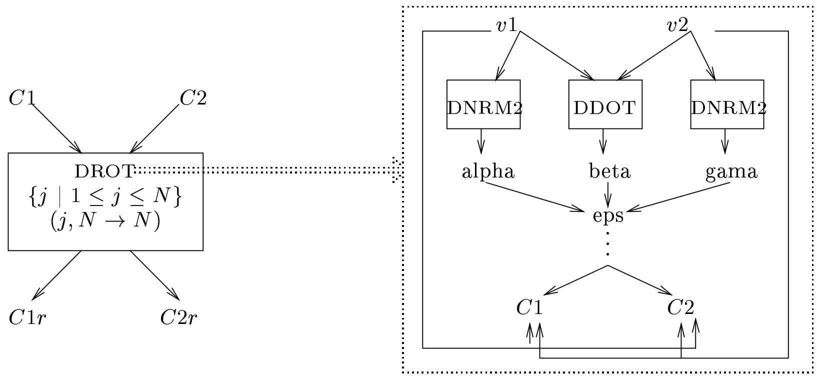

Fig. 8. Structured dependence graph of the `STEP` and `DROT` systems