Revisiting the decomposition of Karp, Miller and Winograd

Alain Darte and Frédéric Vivien *
LIP, CNRS URA 1398
Ecole Normale Supérieure de Lyon
Lyon, France
e-mail: [darte, fvivien]@lip.ens-lyon.fr

Abstract

This paper is devoted to the construction of multi-dimensional schedules for a system of uniform recurrence equations. We show that this problem is dual to the problem of computability of a system of uniform recurrence equations. We propose a new study of the decomposition algorithm first proposed by Karp, Miller and Winograd: we base our implementation on linear programming resolutions whose duals give exactly the desired multi-dimensional schedules. Furthermore, we prove that the schedules built this way are optimal up to a constant factor.

1: Introduction

This paper is concerned with the problem of building multi-dimensional schedules for system of uniform recurrence equations, especially when there exists no linear schedule.

Systems of uniform recurrence equations were first defined and studied, by Karp, Miller and Winograd, in the now famous paper: "The organization of computations for uniform recurrence equations" [6]. This model happened to be powerful enough to describe a large class of algorithms, and simple enough to allow a precise description of the organization of computations. These two properties are certainly responsible for the success of uniform recurrence equations over the last decade, especially in the field of systolic array design methodologies [8, 11], for which this model provides a sound mathematical framework.

In [6], Karp, Miller and Winograd described the structure of a system of uniform recurrence equations with a dependence graph, whose edges are labelled with multi-dimensional vectors. They studied the calculability of such systems and showed that this problem is equivalent to the problem of detecting cycles, in the dependence graph, whose total weight is a non positive vector.

When the system of equations is reduced to a single equation, the problem of computability is quite simple and is equivalent to the existence of a linear schedule. This result is the base of most systolic array design methodologies, such as space-time mapping (first introduced in [10] and developed in many other papers), and some parallelization techniques such as the hyperplane method [9, 3].

In the general case, thus the case of several equations, Karp, Miller and Winograd showed that the problem of calculability is decidable and they gave an algorithm that solves it. This algorithm is based on a recursive decomposition of the dependence graph. Deeper studies and variants of this algorithm have been proposed by Rao [12], Kosaraju and Sullivan [7], Cohen and Meggido [2], Backes [1]. All these works deal with cycle detection or longest dependence paths construction.

*Supported by the French Council for Research CNRS, and by the ESPRIT Basic Research Action 6632 "NANA2" of the European Economic Community.

1063-6862/95 $4.00 © 1995 IEEE

However, the dual aspect of this decomposition has never been really studied. In the case of a single equation, it is well-known that studying the dual problem leads to the construction of linear schedules. In the general case, things are much more complicated: we show in this paper that the dual problem leads to the construction of multi-dimensional schedules. The interest of our approach is threefold:

- Any computable system of uniform recurrence equations admits a explicit order of computations, given by a multi-dimensional schedule.
- Such a schedule can be efficiently built by a dual representation of Karp, Miller and Winograd's decomposition.
- Furthermore, we prove that the schedule we build is nearly optimal: it expresses the maximal parallelism implicitly contained in the system of uniform recurrence equations.

In section 2, we introduce the notations and definitions that we need. In section 3, we recall Karp, Miller and Winograd's decomposition that detects null weight cycles in a dependence graph. In section 4, we explain how a step of the decomposition can be efficiently implemented by a single linear programming resolution and we study the dual linear program. In section 5, we show how this dual resolution allows us to build nearly latency optimal multi-dimensional schedules. Finally, in section 6, we give some concluding remarks: we summarize the paper and we show briefly how this work is linked to our first motivation: the automatic parallelization of nested loops. However, due to the lack of space, we refer to [4] for complete proofs and for details on the applications and interests of our approach, especially how it applies to the parallelization of arbitrary nested loops and how to rewrite the code to handle "shifted-linear" schedules.

2: Definitions and notations

$\mathbb{Z}$ and $\mathbb{Q}$ denote, respectively, the set of integral and rational numbers, and $\mathbb{Z}^n$ and $\mathbb{Q}^n$ denote the set of n-dimensional integral and rational vectors. $[z]_i$ denotes the i-th component of vector z, and $x.y$ the dot product of vectors x and y. For any finite set S, we denote by $\#S$ the number of elements in S.

2.1: Systems of uniform recurrence equations (SURE)

Definition by equations: A **system of uniform recurrence equations** (SURE) is a finite set of equations of the form:

$$V_i(z) = f_i(V_{i_1}(z - d_{i_1,i}), \ldots, V_{i_{m_i}}(z - d_{i_{m_i},i}))$$

- V_i is a data array (or **variable**) to be computed for all integral points z included in a subset $\mathcal{P}$ of $\mathbb{Z}^n$. $\mathcal{P}$ is called the **iteration domain**. Boundary values are assumed to be given at points $z \notin \mathcal{P}$ wherever required for variable evaluation at points $z \in \mathcal{P}$.
- z is a vector in $\mathbb{Z}^n$ called **iteration vector**.
- $d_{i,j}$ are vectors in $\mathbb{Z}^n$ called **dependence vectors**.
- f_i is a strict function with m_i arguments whose properties (if any) will not be considered.

One says that $V_i(z)$ depends on $V_{i_1}(z - d_{i_1,i})$, ..., $V_{i_{m_i}}(z - d_{i_{m_i},i})$. These dependence relations define a graph in $\mathcal{P} \subset \mathbb{Z}^n$ called the **expanded dependence graph**.

Example: Throughout the paper, we will work on the following example:

$$\begin{cases} a(i,\ j,\ k) = b(i+1,\ j-1,\ k) + a(i,\ j,\ k-1) \\ b(i,\ j,\ k) = a(i-1,\ j,\ k) + b(i,\ j,\ k+1) \end{cases}$$

Definition by dependence graph: A system of uniform recurrence equations can be defined in terms of a multigraph G called a **reduced dependence graph** (or briefly, dependence graph) that gives a condensed representation of the expanded dependence graph. G is naturally defined as follows:

- For each variable V_i, there is a vertex v_i in G.
- For each V_i such that $V_i(z)$ depends on $V_j(z - d_{j,i})$, there is an edge in G, from v_j to v_i labelled with vector $d_{j,i}$. Such a label is called the **weight** of the edge.

A system of uniform recurrence equations is then simply defined by a dependence graph G and an iteration domain $\mathcal{P}$. In the following, V denotes the set of vertices of G, E the set of edges of G. $w(e)$ represents the weight of an edge e (i.e. the dependence vector associated to it), $t(e)$ the vertex from which edge e is directed (the tail) and $h(e)$ the vertex to which edge e is directed (the head).

Example: The dependence graph G associated to our example is given in figure 1.

Difference between SUREs and nested loops: A SURE may seem identical to perfectly nested loops with uniform dependences, since both can be defined by a dependence graph with uniform dependences and an iteration domain. However, the way these graphs are built makes a huge difference.

In nested loops, there is an **explicit** order of computations: the lexicographic order of iteration vectors, since innermost loops are scanned for each scan of the outermost loops. Dependence vectors can be flow, anti or output dependences but they are always lexicographically positive: therefore, the expanded dependence graph is acyclic and the reduced dependence graph has no cycle of null weight.

In a SURE, the order of computations is an **implicit** order: the left-hand side of each equation can be computed only if all arguments of the right-hand side have been computed. All dependence vectors can be seen as flow dependences, since values are used after being computed, but they do not have to be lexicographically positive. The main consequence is that a SURE is not always computable.

In this sense, the framework of SURE is a more general framework than perfectly nested loops with uniform dependences. The simple fact that the dependence vectors do not have to be lexicographically positive leads to two difficult problems:

i. A SURE is not always computable: this problem is linked to the problem of detecting null weight cycles in the reduced dependence graph G [1].

ii. The order of computations is implicit: building an explicit order is linked to the construction of multi-dimensional schedules as defined in section 2.2.

The goal of this paper is to show that these two problems are actually dual problems and that they can be solved efficiently by an algorithm based on a recursive decomposition of the dependence graph G.

[1] This was the main result of Karp, Miller and Winograd: the computability of a SURE is decidable (and is in P). However, for all known extensions of the model of SURE, the problem is undecidable (see [14]).

2.2: Schedules

A schedule gives an explicit order of the computations described by a SURE. Among all schedules, schedules that correspond to loop executions are particularly interesting. A linear schedule (wave-front or hyperplane method [9, 3]), defined below, corresponds in term of loops to the possibility of rewriting a SURE with one outermost sequential loop and all parallel inner loops. However, such a schedule does not exist for all SUREs. When the degree of parallelism contained in the SURE is not sufficient, one has to consider an extension of the notion of schedule, called multi-dimensional schedule.

One-dimensional schedules: A **one-dimensional schedule** is a mapping from $V \times \mathbb{Z}^n$ to $\mathbb{Z}$ that gives an execution order to the computation of each value $V_i(z)$ while preserving dependence relations. Thus, a function T, with values in $\mathbb{Z}$, is a schedule if and only if for all iteration vectors p and q, for all variables v and w:

$$T(v,p) \geq T(w,q) + 1 \text{ whenever } (w,q) \rightarrow (v,p) \tag{1}$$

where $(w,q) \rightarrow (v,p)$ means that $V(p)$ depends on $W(q)$ [2].

Note that if T satisfies inequalities 1 but takes its values in $\mathbb{Q}$ instead of $\mathbb{Z}$, then $\lfloor T \rfloor$ is a schedule since

$$T(v,p) \geq T(w,q) + 1 \Rightarrow T(v,p) \geq \lfloor T(w,q) \rfloor + 1 \Rightarrow \lfloor T(v,p) \rfloor \geq \lfloor T(w,q) \rfloor + 1$$

For this reason, in the rest of the paper, we will be interested in rational functions that satisfy inequalities 1, since they are easier to build. We will call such functions rational schedules.

One way of measuring the performance of a schedule T is its **latency** $L(T,\mathcal{P})$, i.e the total number of computation steps needed to execute the SURE on its iteration domain $\mathcal{P}$ according to the schedule T: $L(T,\mathcal{P}) = \#\{T(V \times \mathcal{P})\}$.

Among all one-dimensional schedules, some interesting classes of schedules are linear schedules, shifted-linear schedules and affine schedules: a schedule is an **affine schedule** if it is a mapping T of the following form (where $X_v \in \mathbb{Q}^n$ and $\rho_v \in \mathbb{Q}$):

$$\begin{array}{rcl} T: V \times \mathbb{Z}^n & \longrightarrow & \mathbb{Q} \\ (v,\ p) & \rightarrow & X_v.p + \rho_v \end{array}$$

If all X_v are equal, T is called a **shifted-linear schedule**, and if, moreover, all ρ_v are null, then T is called a **linear schedule**. Note that for shifted-linear schedules, inequalities 1 reduces to:

$$\forall e \in G,\ X.w(e) + \rho_{h(e)} - \rho_{t(e)} \geq 1$$

Such a vector X is called a **strictly separating hyperplane**. When we just have the relation $X.w(e) + \rho_{h(e)} - \rho_{t(e)} \geq 0$, X is said to be a **weakly separating hyperplane** for edge e.

Multi-dimensional schedules: We are now ready to define multi-dimensional schedules introduced by Karp, Miller and Winograd [6], Rao [12] in the context of SURE and Feautrier [5] in the context of nested loops.

[2] this notation implicitly implies that $p, q \in \mathcal{P}$.

A **multi-dimensional schedule** (of dimension d) is a mapping from $V \times \mathbb{Z}^n$ to $\mathbb{Z}^d$ that gives an execution order to the computation of each value $V_i(z)$, while preserving dependence relations.

The execution order is now a d-dimensional order, that we take to be the strict lexicographic order $\gg_l$ (so as to correspond to the execution of d nested loops): you can visualize this by saying that the successive components given by the schedule correspond to days, hours, minutes, seconds, and so on ...

The compatibility with dependence relations is not an inequality on values anymore, but on d-dimensional vectors and is expressed with $\gg_l$. The constraints to be satisfied now are:

$$T(v,p) \gg_l T(w,q) \text{ whenever } (w,q) \rightarrow (v,p) \tag{2}$$

that is to say:

$$\exists k,\ 1 \leq k \leq d, \forall i,\ i < k,\ [T(v,p)]_i = [T(w,q)]_i \text{ and } [T(v,p)]_k \geq [T(w,q)]_k + 1 \tag{3}$$

whenever $(w,q) \rightarrow (v,p)$. As previously, integral schedules are extended to rational functions that satisfy constraints 3.

Once again, things get simpler when considering particular schedules such as affine multi-dimensional schedules defined as follows: a d-dimensional schedule is an **affine schedule** if it is a mapping T of the following form (where $X_v^i \in \mathbb{Q}^n$ and $\rho_v^i \in \mathbb{Q}$):

$$\begin{array}{rcl} T:\ V \times \mathbb{Z}^n & \longrightarrow & \mathbb{Q}^d \\ (v,\ p) & \rightarrow & (X_v^1.p + \rho_v^1, \ldots, X_v^d.p + \rho_v^d) \end{array}$$

When all vectors X_v^i are equal, inequalities 3 reduce to:

$$\begin{cases} \forall e \in E,\ \exists k_e,\ 1 \leq k_e \leq d & \text{such that} \\ \forall i,\ 1 \leq i < k_e & X^i.w(e) + \rho^i_{h(e)} - \rho^i_{t(e)} = 0 \\ & X^{k_e}.w(e) + \rho^{k_e}_{h(e)} - \rho^{k_e}_{t(e)} \geq 1 \end{cases}$$

Thus, the existence of such multi-dimensional schedules is strongly related to the notion of weakly and strictly separating hyperplanes. The objective of this paper is to build such schedules and to show the link between their dimension and the parallelism contained in the SURE.

3: Detection of null weight cycles in a SURE

Consider a system of uniform recurrence equations defined by its iteration domain $\mathcal{P}$ and its reduced dependence graph G. When $\mathcal{P}$ is bounded, the system is computable if and only if the expanded dependence graph has no cycle. When the SURE is not computable, there is a cycle in the expanded dependence graph: therefore, G has a cycle of null weight. Conversely, if G has a null weight cycle, one can build a dependence cycle in $\mathcal{P}$ - if $\mathcal{P}$ is sufficiently large so that boundary problems can be avoided.

Therefore, we will assume in the following that $\mathcal{P}$ is bounded and sufficiently large, so that the SURE is computable if and only if G has no cycle of null weight.

In this section, we recall how the problem of detecting null weight cycles in a reduced dependence graph can be solved, by an algorithm first proposed by Karp, Miller and Winograd [6], then studied among others by Rao [12], Kosaraju and Sullivan [7], Cohen and Meggido [2]. We also give an interpretation of the number of recursive calls in the algorithm: we show that it is linked to the inherent sequentiality contained in the SURE.

3.1: Definitions and notations

Matrix notations: We denote by C the **connection matrix** of the reduced dependence graph G, defined as follows: each row of C corresponds to a vertex of G and each column of C to an edge of G. If the j-th edge of G is oriented from vertex i to vertex k, then we let $C_{i,j} = -1$ and $C_{k,j} = +1$, otherwise for all $l \notin \{i, k\}$, we let $C_{l,j} = 0$. A special case exists for self-dependence edges for which we let $C_{l,j} = 0$, for all l.

We denote by D the **dependence matrix** whose columns are the dependence vectors. Of course, edges of G are considered in the same order for D and for C.

Finally, we denote by B, the block matrix whose first rows are the rows of C and whose last rows are the rows of D.

Example: For our example (given by figure 1) with 2 vertices and 4 edges, we have:

$$C = \begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} C \\ D \end{bmatrix}$$

Subgraph of null weight multi-cycles: The previous matrix notations allow us to formulate the notion of multi-cycle of null weight. A multi-cycle is a union of cycles, not necessarily connected. If there exists a vector q (with non negative integer components) such that $Cq = 0$, then G has a multi-cycle that uses q_i times the i-th edge of G. Moreover, if $Dq = 0$ (thus $Bq = 0$), then the multi-cycle is a multi-cycle of null weight. Thus, detecting a null weight multi-cycle can be easily done by checking if the following system has a rational solution:

$$\{\ q \geq 0, \quad q \neq 0, \quad Bq = 0\ \} \tag{4}$$

(If there exists a rational solution, then by scaling its components, there exists an integral solution).

However, detecting a null weight cycle is much more difficult. Every null weight cycle is a null weight multi-cycle but the converse is not necessarily true. However, when all edges of a null weight multi-cycle form a strongly connected graph, a null weight cycle can be found as will be stated in lemma 1. This is the underlying idea of the decomposition algorithm of Karp, Miller and Winograd.

Let G' be the subgraph of G induced (by the residual edges), after all edges in G that belong to any null weight multi-cycles have been deleted. In [6], this is achieved by testing, for each edge e_i, whether the system 4 has a feasible solution with $q_i > 0$. We will show in section 4 that G' can be built more efficiently by solving only one linear program.

3.2: Decomposition algorithm

Determining if a SURE is computable can be done by applying the following algorithm to its dependence graph G.

Algorithm:

i. Decompose G into strongly connected components $G_1, G_2, \ldots, G_s$, and call step (ii) on each G_i.

ii. Build G' the subgraph of G generated by all edges that belong to a null weight multi-cycle of G.

- If G' is strongly connected then G is not computable.
- If G' is an empty graph, G is computable.
- Otherwise call step (i) on G'.

We will see in section 5.3 how the decomposition algorithm works on our example.

The correctness of the algorithm is based on several lemmas, proved in [6] and reformulated in [4]. The most important lemma is the following one:

Lemma 1 (Karp, Miller and Winograd) *If the subgraph G' of a graph G is strongly connected, G has a null weight cycle.*

Corollary 1 *The decomposition algorithm is correct.*

Proof: Note first that the decomposition algorithm always ends, when applied to a finite graph. When step (ii) is applied to a graph G, either the algorithm ends because G' is strongly connected or null, or step (ii) is called on strictly **smaller** subgraphs of G (the strongly connected components of G').

If the decomposition ends because one of the G'_i is strongly connected, then G'_i has a null weight cycle as shown by lemma 1 and so does G. The corresponding SURE is not computable. On the other hand, if all recursive calls end because the corresponding subgraphs G'_i are empty, G has no null weight cycle and the corresponding SURE is computable. □

We denote by d the **depth** of the decomposition algorithm, i.e. the longest sequence of recursive calls, except if all vertices are "lonely" (in the sense that no edge passes through them) in which case we let $d = 0$.

4: Construction and properties of G'

In this section, we show how to build G' efficiently (by one linear programming problem instead of one per edge as in [6]). Furthermore, we show that this construction is closely related to the construction of weakly and strictly separating hyperplanes and thus to affine multi-dimensional schedules.

4.1: Linear program for building G'

Step (ii) of the decomposition algorithm, which consists in the construction of G', can be done by solving only one linear program. We will show indeed that the edges of G' are exactly the edges e_i for which $v_i = 0$ in any optimal solution of linear program 5.

$$\min \{ \ \textstyle\sum_i v_i \ \mid \ q \geq 0, \quad v \geq 0, \quad q + v \geq 1, \quad Bq = 0 \ \} \tag{5}$$

Note first that linear program 5 has a finite solution: $q = 0$ with $v = 1$ is indeed a solution.

Lemma 2 *For any optimal solution (q, v) of program 5:*

- $q_i \neq 0 \Leftrightarrow v_i = 0$.
- $q_i = 0 \Leftrightarrow v_i = 1$.
- $v_i = 0 \Leftrightarrow e_i \in G'$

Therefore, solving program 5 gives directly G': the edges of G' are the edges e_i such that $v_i = 0$.

4.2: Interpretation of the dual

Now, to better understand what is behind this linear program, let us consider its dual. Program 5 can be written in a canonical form as:

$$\min \{ \sum_i v_i \mid q \geq 0, \quad v \geq 0, \quad w \geq 0, \quad q + v = 1 + w, \quad Bq = 0 \} \tag{6}$$

Its dual can be written (after some manipulations) as:

$$\max \{ \sum_i z_i \mid z \geq 0, \quad 0 \leq z_i \leq 1, \quad X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} \geq z_i \} \tag{7}$$

where inequality $z_i \geq 0$ corresponds to variable w_i, inequality $z_i \leq 1$ to variable v_i while inequality $X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} \geq z_i$ corresponds to variable q_i. The dual solution has an interesting property as shown by the following lemma.

Lemma 3 *For any optimal solution (z, X, ρ) of the dual program 7:*

$$e_i \in G' \Leftrightarrow X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} = 0 \tag{8}$$

$$e_i \notin G' \Leftrightarrow X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} \geq 1 \tag{9}$$

Lemma 3 shows that considering the dual provides separating hyperplanes, which are strictly separating hyperplanes for the edges not in G' and weakly separating hyperplanes for edges in G'. Furthermore, for each subgraph G that appears in the decomposition, these hyperplanes are those that are the "most often strict": the number of edges, for which such an hyperplane is strict, is maximal.

During the decomposition algorithm, one can associate to each vertex v of G, a sequence of vectors $X_v^1, \ldots, X_v^{d_v}$, obtained by considering the dual problem 7. d_v is the depth of the decomposition algorithm at which vertex v is removed. This sequence of vectors has the following property:

Theorem 1 *The $d_v - 1$ first separating hyperplanes $X_v^1, \ldots, X_v^{(d_v-1)}$ associated to a vertex v of G are linearly independent. Furthermore, when G is computable, all separating hyperplanes $X_v^1, \ldots, X_v^{d_v}$ associated to v are linearly independent.*

Note that this implies that the depth of the decomposition is bounded by $n + 1$ (and even by n when G is computable). This permits to give an upper bound on the time complexity of the decomposition algorithm (see [4]).

Remark: in practice, linear programs 5 and 7 can be simplified by replacing $Cq = 0$ by $q = \mu_1 q_1 + \ldots + \mu_m q_m$ where $q_1, \ldots, q_m$ form a basis of cycles. This reduces the number of inequalities in program 5 and the number of variables in program 7 (constants ρ disappear in this new formulation). The constants ρ can then be computed by an algorithmic approach, less expensive than a linear programming resolution, simply by computing the longest paths in a graph similar to G but where edge e_i has a weight equal to $z_i - X.w(e_i)$ (Bellman-Ford algorithm). This is roughly the way we implemented it.

5: Optimal multi-dimensional schedule for a computable SURE

We are now ready to use the dual interpretation of Karp, Miller and Winograd's algorithm for building nearly optimal multi-dimensional schedules.

5.1: Construction of a d-dimensional schedule

Consider a computable dependence graph G and assume first that G is strongly connected. We apply to G the decomposition algorithm, focusing this time on the dual program 7.

We build for each vertex v of G, the sequence of vectors $X_v^1, \ldots, X_v^{d_v}$ and the sequence of constants $\rho_v^1, \ldots, \rho_v^{d_v}$, obtained by considering the dual program 7 during the decomposition algorithm. For each vertex v, we complete the sequences of vectors X_v^i and constants ρ_v^i with zeros so as to obtain sequences of length d. This defines a function T:

$$\begin{array}{rcl} T: V \times \mathcal{P} & \longrightarrow & \mathbb{Q}^d \\ (v,\ p) & \rightarrow & (X_v^1.p + \rho_v^1,\ \ldots,\ X_v^{d_v}.p + \rho_v^{d_v},\ 0,\ \ldots,\ 0) \end{array}$$

Lemma 4 *T defines a multi-dimensional schedule.*

Proof: We just have to show that for all edges e, for all $p \in \mathcal{P}$:

$$T(h(e), p) \gg_l T(t(e), p - w(e))$$

G is computable, thus the decomposition algorithm ended because all the leaves of the calling tree ended with an empty G'. Thus, at some level of the decomposition, edge e has been removed, i.e. $e \in G_k$ but $e \notin G'_k$. By construction, until level k, $h(e)$ and $t(e)$ belong to the same subgraph of G, thus their sequences of vectors X^i are the same until level k:

$$X^1_{h(e)} = X^1_{t(e)} = X^1,\ \ldots,\ X^k_{h(e)} = X^k_{t(e)} = X^k \tag{10}$$

Furthermore, until level $k-1$, vectors X^i are weakly separating hyperplanes for edge e and at level k, since e has been removed, X^k is a strictly separating hyperplane for edge e. Thus, the i-th component ($i < k$) of $T(h(e), p) - T(t(e), p - w(e))$ is equal to:

$$\left(X^i.p + \rho^i_{h(e)}\right) - \left(X^i.(p - w(e)) + \rho^i_{t(e)}\right) = X^i.w(e) + \rho^i_{h(e)} - \rho^i_{t(e)} = 0$$

and the k-th component satisfies:

$$\left(X^k.p + \rho^k_{h(e)}\right) - \left(X^k.(p - w(e)) + \rho^k_{t(e)}\right) = X^k.w(e) + \rho^k_{h(e)} - \rho^k_{t(e)} \geq 1$$

Finally, whatever are the rests of the sequences $X^i_{h(e)}$, $X^i_{t(e)}$, $\rho^i_{h(e)}$ and $\rho^i_{t(e)}$ after level k, one has $T(h(e), p) \gg_l T(t(e), p)$ for all $p \in \mathcal{P}$. The dependence corresponding to edge e is satisfied at level k. □

Remarks:

- The separating hyperplanes X that define T are obviously not unique: once we know which edge corresponds to an equality like 8 and which one corresponds to an inequality like 9, one can choose another objective function than $\sum_i z_i$. One can for example try to minimize the latency corresponding to the vector X, i.e. $\max_{p \in \mathcal{P},\ q \in \mathcal{P}} X.(p-q)$. See [3] for this kind of optimization.

- The multi-dimensional schedules so obtained are not arbitrary schedules: they are indeed at each level affine schedules whose linear part is the same on the current graph G_k (see equation 10). We call them, **locally** shifted-linear schedule. This is an important property in practice, because it does not generate too complicated results, that would be difficult to use.

When the dependence graph has more than one strongly connected component, we first schedule independently each strongly connected component with a multi-dimensional schedule built as above. Then, we schedule them with respect to each other by a topological sort on the acyclic graph constitued by the strongly connected components.

5.2: Longest dependence path in a SURE and latency of the multi-dimensional schedules

Once we know that a system of uniform recurrence equations is computable or not, it is interesting to give an idea of the length of the longest path in the expanded dependence graph. This length gives a lower bound on the sequentiality of the system of recurrence equations and thus gives also an upper bound on the parallelism it contains.

In [3], it is shown that for **a single** uniform recurrence equation, this length is equivalent to the latency of the optimal linear schedule, on full dimensional polyhedra [3] whose size tends to infinity. To say it briefly, on domains of size parameterized by N, the latency of the optimal linear schedule is equivalent to λN for some constant λ and so does the length of the longest dependence path. Both are in N and the multiplicative constants λ are the same.

Here, in a SURE, when linear schedules do not always exist, the length of the longest path is not necessarily linear in N anymore, it can be equivalent to kN^p for some constants k and p. We will not try to be precise on the multiplicative constant k, we will just focus on p, the power of N.

Consider a SURE defined in $\mathbb{Z}^n$ by an iteration domain $\mathcal{P}$ and a dependence graph G. Suppose that $\mathcal{P}$ contains a n-dimensional cube of size N (thus $\mathcal{P}$ is full dimensional) and is contained in a n-dimensional cube of size λN, for some constant $\lambda \geq 1$. With these hypothesis, the link between the latency of the multi-dimensional schedules we built, and the length of the longest dependence path is given by the following theorem:

Theorem 2 *The multi-dimensional schedule built in section 5 is nearly optimal: if d is the depth of the decomposition algorithm, the latency of the schedule is $O(N^d)$ and the length of the longest dependence path is $\Omega(N^d)$.*

This means that the system of uniform recurrence equations corresponding to G contains a parallelism of degree $(n-d)$ and that we are able to find it.

5.3: Example

We now go back to our example (given in section 2.1).

Decomposition: In practice, to check the computability of the studied SURE, we need to apply the decomposition algorithm of subsection 3.2 and thus to solve the following linear program (linear program 5):

$$\min \{ \ \textstyle\sum_i v_i \ \mid \ q \geq 0, \quad v \geq 0, \quad q + v \geq 1, \quad Bq = 0 \ \}$$

[3] see [13] for a definition of the dimension of a polyhedron

$$\text{With } q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix},\ Bq = 0 \Leftrightarrow \begin{cases} -q_2 + q_4 = 0 \\ q_2 - q_4 = 0 \\ -q_2 + q_4 = 0 \\ q_2 = 0 \\ -q_1 + q_3 = 0 \end{cases}. \text{ Thus } \exists \lambda \geq 1,\ q = \lambda * \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix},\ v = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

This means that only the self-dependences are members of G', the subset of the edges of G which can participate to a null-weight multi-cycle. The same study is made on the two strongly connected components of G'. Each one contains one of the two self-dependences of G. As none of these self-dependences is null, none of the connected components of G' contains a null-weight multi-cycle. The theoretical results previously presented let us conclude that the studied system of uniform recurrence equations is computable. As the decomposition algorithm stops at depth 2, there exists a multi-dimensional schedule whose latency is $O(N^2)$ and a dependence path of length $\Omega(N^2)$.

The decomposition is illustrated by the figure 1.

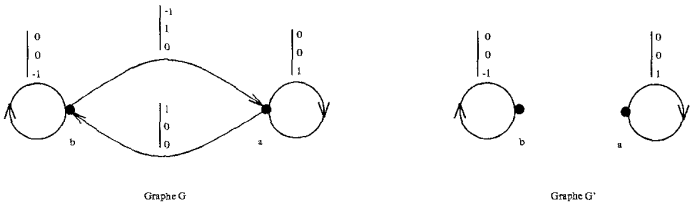

Figure 1. Decomposition of the dependence graph

Multi-dimensional schedule: Hence, we are assured of the computability of the SURE. We now have to build **a** multi-dimensional schedule, and thus to solve the linear program 7. In our example, we choose as an objective function to minimize $\sum_i |X_i|$, the norm of the vector X so as to get not too complicated results. Note that if the iteration domain is a cube, this is equivalent to minimizing the latency.

Now, because of lemma 3, we have to find a solution to the system:

$$\min \left\{ \sum_i |X_i| \;\middle|\; \begin{array}{l} e_i \in G' \Rightarrow X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} = 0 \\ e_i \notin G' \Rightarrow X.w(e_i) + \rho_{h(e_i)} - \rho_{t(e_i)} \geq 1 \end{array} \right\}$$

If we note $X = [x,\ y,\ z]$, this system can be re-written as follows:

$$\min \left\{ |x| + |y| + |z| \;\middle|\; \begin{array}{r} -x + y + \rho_a - \rho_b \geq 1 \\ x + \rho_b - \rho_a \geq 1 \\ -z = 0 \\ z = 0 \end{array} \right\}$$

The optimal solutions of this linear program is: $X = [0,\ 2,\ 0], \rho_a = 0,\ \rho_b = 1$.

To this point, we have solved the problem on two of the four edges. We have then to work recursively on the remaining strongly connected components of G', but in this

case these components are very simple and the linear programs are trivial: for the connected component which only contains the dependence [0, 0, 1], an optimal solution is $X = [0,\ 0,\ 1]\,, \rho_a = 0$, and for the connected component which only contains the dependence [0, 0, −1], an optimal solution is $X = [0,\ 0,\ -1]\,, \rho_b = 0$.

The final result is thus the following:

variable	schedule	first level	second level
a			
	linear parts	[0, 2, 0]	[0, 0, 1]
	constant parts	0	0
b			
	linear parts	[0, 2, 0]	[0, 0, −1]
	constant parts	1	0

and, if $\mathcal{P}$ is the 3-dimensional cube of size N, it leads to the following nested loops:

```
for j = 1 to N          (vector (0,2,0), constant 0)
  for k = 1 to N          (vector (0,0,1))
    forall i = 1 to N
      a(i, j, k) = b(i+1, j-1, k) + a(i, j, k-1)
    endforall
  endfor
                          (vector (0,2,0), constant 1)
  for k = N to 1          (vector (0,0,-1))
    forall i = 1 to N
      b(i, j, k) = a(i-1, j, k) + b(i, j,  k+1)
    endforall
  endfor
endfor
```

6: Conclusion

The results presented in this paper are mainly based on Karp, Miller and Winograd's decomposition algorithm which decides if a system of uniform recurrence equations is computable or not. We first showed how this algorithm can be efficiently implemented by linear programming resolutions. Then, by an interpretation of the dual linear programs, we explained how to build multi-dimensional schedules that express explicit orders of computations for a computable system of uniform recurrence equations.

The first interest of this resolution is that we are able to guarantee the quality of such schedules: they are indeed nearly optimal in the sense that their latency and the length of the longest dependence paths are of the same order. The schedules that we obtain reveal the maximal degree of parallelism contained in the SURE, while the longest dependence paths reveal the maximal degree of sequentiality in the SURE.

The second interest, and in our opinion not the less important, is that such schedules have a very particular structure: we called them locally shifted-linear schedules. That means that there is no need to look for complicated schedules such as arbitrary affine multi-dimensional schedules. The resolution gives directly the simpliest form that can be considered to keep optimal latency. The simplicity of the results has to be taken into account, since the final goal is to use such schedules to generate code. The simpler the schedule, the easier the implementation, the better the chance of further optimizations.

Because of space limitation, we did not address in this paper the possible applications of this construction of schedules. Actually, the initial motivation of this work was not in the context of systems of uniform recurrence equations but in the context of nested loops. Our goal was to find a way to parallelize arbitrary nested loops, especially when dependences are not uniform but are expressed as direction vectors or dependence cones: our idea was to try to extend linear scheduling methods. Finally, we found much more than what was expected: based on this study of Karp, Miller and Winograd's decomposition, we are now able to propose a new and optimal parallelization technique for nested loops. The main idea is to transform the dependence graph associated to the nested loops into an equivalent dependence graph where all dependences are uniform but not necessarily lexicographically positive, i.e. nothing but the dependence graph of a SURE on which we apply the techniques presented in this paper. We refer to [4] for more details.

We believe that this approach is a very promising approach since it can be proven optimal with respect to the quality of the dependence analysis.

Dedication We would like to dedicate this work to Hervé Le Verge who died a few months ago. Hervé was a bright researcher in the field of systolic methodology design.

References

[1] Wolfgang Backes. *The structure of longest paths in periodic graphs.* PhD thesis, Universität des Saarlandes, Sarbrücken, July 1993.

[2] Edith Cohen and Nimrod Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs. In *Prodceedings of 21st Annual ACM Symposium on Theory of Computing*, pages 523–534, 1989.

[3] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[4] Alain Darte and Frédéric Vivien. Automatic parallelization based on multi-dimensional scheduling. Technical Report 94-24, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, September 1994.

[5] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II, multi-dimensional time. *Int. J. Parallel Programming*, 21(6):389–420, December 1992. Available as Technical Report 92-78, Laboratoire MASI, Université Pierre et Marie Curie, Paris, October 1992.

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

[7] S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In ACM Press, editor, *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 398–406, May 1988.

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

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

[10] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *The 11th Annual International Symposium on Computer Architecture*, Ann Arbor, Michigan, June 1984. IEEE Computer Society Press.

[11] Patrice Quinton and Yves Robert. *Systolic Algorithms and Architectures.* Prentice Hall, 1991. Translated from French, Masson (1989).

[12] Sailesh K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, October 1985.

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

[14] Hervé Le Verge and Yannick Saouter. New results on computability of recurrence equations. Technical Report 380-93, LaBRI, Bordeaux (France), September 1993. Submitted to Foundations of Computer Science.