Some efficient solutions to the affine scheduling problem
Part II
Multidimensional time

Paul Feautrier

Laboratoire MASI,
Institut Blaise Pascal
Université de Versailles Saint-Quentin
45 Avenue des Etats-Unis
78035 VERSAILLES CEDEX FRANCE

e-mail : `feautrier@masi.ibp.fr`

October 7, 1992

Abstract

This paper extends the algorithms which were given in Part I to cases in which there is no affine schedule, i.e. to problems whose parallel complexity is polynomial but not linear. The natural generalization is to multidimensional schedules with lexicographic ordering as temporal succession. Multidimensional affine schedules, are, in a sense, equivalent to polynomial schedules, and are much easier to handle automatically. Furthermore, there is a strong connexion between multidimensional schedules and loop nests, which allows one to prove that a static control program always has a multidimensional schedule. Roughly, a larger dimension indicates less parallelism. In the algorithm which is presented here, this dimension is computed dynamically, and is just sufficient for scheduling the source program. The algorithm lends itself to a "divide and conquer" strategy. The paper gives some experimental evidence for the applicability, performances and limitations of the algorithm.

Résumé

Dans cet article, les algorithmes qui ont été proposé dans la première partie sont étendus au cas où le programme source n'a pas de base de temps affine, c'est-à-dire à des algorithmes dont la complexité parallèle est polynomiale mais non linéaire. La solution naturelle est l'emploi de bases de temps à plusieurs dimensions, l'ordre de succession temporelle étant l'ordre lexicographique. Les bases de temps multidimensionnelles sont, en un certain sens, équivalentes à des bases de temps polynomiales, et sont beaucoup plus faciles à manipuler algorithmiquement. De plus, il y a une connexion forte entre bases de temps multidimensionnelles et nids de boucles, ce qui permet de démontrer qu'un programme à contrôle statique a toujours une base de temps multidimensionnelle. En gros, plus grande est la dimension et moins il y a de parallélisme. Dans l'algorithme ici présenté, cette dimension est déterminée dynamiquement; elle est juste suffisante pour permettre l'ordonnancement du programme source. Enfin, cet algorithme se prête à l'application de la stratégie "diviser pour régner". On présente en conclusion quelques résultats expérimentaux permettant de juger du domaine d'application, des performances et des limitations de l'algorithme.

```
        do i = 0,n
           do j = 0,i
1             s = s + a(i,j)
           end do
        end do
```

Figure 1: A simple program with no linear schedule

Instruction	Domain		
1	$0 \leq i \leq n$ $0 \leq j \leq i$		
Edge	Source	Destination	Condition
1	$(1, i, j-1)$	$(1, i, j)$	$j \geq 1$
2	$(1, i-1, i-1)$	$(1, i, j)$	$j < 1 \wedge i \geq 1$

Table 1: The DFG of program 1

1 Introduction

In the first part of this paper [1], I have presented a new algorithm for computing affine and piecewise affine schedules for Generalized Dependence Graphs and affine systems of recurrence equations. The algorithm is simple and efficient. However, there are programs and systems which do not have such a schedule. This is equivalent to the observation that there are programs which cannot be executed in linear time on a paracomputer, and should not come as a surprise.

> Consider the program in figure 1. For definiteness, suppose that + in this computation stands for some operator with no special algebraic property. As a consequence, the computation must be executed as written; there is no possibility of sharing the work between processors by taking advantage, e.g. of associativity[1].
>
> The DFG of program 1 is given in table 1. The only flow of data in this program is throught the successive values of variable `s`. Each iteration of instruction 1 uses a value of `s` which has been produced by the immediately preceding iteration, which is $\langle 1, i, j-1\rangle$ if $j \geq 1$ and $\langle 1, i-1, i-1\rangle$ if not.
>
> To construct an affine schedule, let us apply the Farkas algorithm. The prototype schedule is:

$$\theta(i, j) = \mu^0 + \mu^1 i + \mu^2(n-i) + \mu^3 j + \mu^4(i-j).$$

[1] This remark will stand for all examples in this paper. Computing schedules for operators with nontrivial algebraic properties is a largely open problem.

Edge 1 is a uniform dependence, which gives simply:

$$\mu^3 - \mu^4 \geq 1. \tag{1}$$

Edge 2 is nonuniform; we must use Farkas lemma. Note that the constraint $i \geq 0$ is redundant in the presence of $i \geq 1$. The result is:

$$\begin{aligned} & \mu^0 + \mu^1 i + \mu^2(n-i) + \mu^3 j + \mu^4(i-j) \\ - & (\mu^0 + \mu^1(i-1) + \mu^2(n-i+1) + \mu^3(i-1)) - 1 \\ \equiv & \lambda^0 + \lambda^1(i-1) + \lambda^2(n-i) + \lambda^3 j + \lambda^4(i-j) + \lambda^5(1-j). \end{aligned}$$

This is equivalent to:

$$\begin{aligned} \mu^1 - \mu^2 + \mu^3 - 1 &= \lambda^0 - \lambda^1 + \lambda^5, & (2)\\ \mu^4 - \mu^3 &= \lambda^1 - \lambda^2 + \lambda^4, & (3)\\ \mu^3 - \mu^4 &= \lambda^3 - \lambda^4 - \lambda^5, & (4)\\ 0 &= \lambda^2. & (5) \end{aligned}$$

These conditions are clearly inconsistent. From (3) and (5) one derives:

$$\mu^4 - \mu^3 = \lambda^1 + \lambda^4 \geq 0,$$

which contradicts (1).

The reader may care to verify that program 1 has the following nonlinear schedule:

$$t(i,j) = \frac{i(i+1)}{2} + j,$$

and that this schedule is in fact optimal. Its latency is:

$$L = \frac{n(n+3)}{2};$$

it is equal to the total operation count. Hence its mean degree of parallelism is 1, i.e. no parallelism at all. Another schedule is:

$$t'(i,j) = ni + j.$$

According to the definitions of this paper, t' is not linear. Neither is it optimal: its mean parallelism is about $\frac{1}{2}$! Note however that for any given value of n, t' *is* linear.

These observations would lead to a search for polynomial schedules. However, polynomials are notoriously difficult to handle in a completely automatic way. Examination of program 1 and other similar examples suggests another solution: the use of multidimensional schedules. In section 2, I will discuss the notion of a multidimensional schedule and state the appropriate version of the causality condition. It is possible to prove that the DFG of a sequential program always has a multidimensional affine schedule. In section 3, I will give the basic algorithm for finding such schedules. This algorithm has a very high complexity; in section 4, I will discuss a much simpler version, which is correct but no longer complete. It is, however, still possible to prove termination in the case of DFG of sequential programs. I will then show how to combine this algorithm with the dualization method of I.3.3.3[2].

In many cases, it is possible to split the scheduling problem into several subproblems according to the strongly connected components (scc) of the dependence graph. This technique is presented in section 5. The algorithm obtained by coupling multidimensional scheduling and scc decomposition may be seen as a generalization of Allen and Kennedy's `codegen` algorithm [6].

Section 6 discusses an implementation of the above algorithm, with respect to the shape of the resulting schedules, completeness and performance. I then discuss some drawbacks of the above methods and give some directions for possible improvements.

2 Multidimensional schedules

2.1 Definition

A schedule is a function which gives the starting time of each operation; time is a quantity whose main property is to belong to a totally ordered set. The schedules which have been constructed up to now have their values in the totally ordered set of natural numbers, but this is not the only possibility. The structure of loop nests suggests the study of schedules whose value is an integral vector, totally ordered by lexicographic ordering. Program 1, for instance, has the following two-dimensional schedule:

$$t''(i,j) = \begin{pmatrix} i \\ j \end{pmatrix}.$$

The first step is to find the correct generalisation of the causality condition:

$$u, v \in E,\ u\Gamma v \Rightarrow t(u) < t(v). \tag{6}$$

An obvious possibility is:

$$u, v \in E,\ u\Gamma v \Rightarrow \theta(u) \ll \theta(v). \tag{7}$$

This is in a sense sufficient, as shown by

[2]References of the form I.3 or (I.12) will designate section or formulas in the first part of this paper.

Theorem 1 *To any d-dimensional schedule θ satisfying (7), one may associate a one-dimensional schedule t, namely:*

$$t(u) = \text{Card}\ \{\theta(x) \mid \theta(x) \ll \theta(u) \wedge x \in E\},$$

which satisfies (6).

Let P_u be the set $\{\theta(x) \mid \theta(x) \ll \theta(u) \wedge x \in E\}$. Let u and v satisfy $u, v \in E$ and $u \Gamma v$, which imply $\theta(u) \ll \theta(v)$. By transitivity,

$$P_u \subseteq P_v.$$

On the other hand, supposing that $\theta(v)$ belongs to P_u would implies a contradiction, since $\ll$ is a strict order. Hence:

$$\text{Card}\ P_u < \text{Card}\ P_v.$$

Note that this proof does not depends on any property of $\ll$ beside its being a strict order. ∎

As a corollary, we get the proper definition of the latency of a multidimensional schedule:

$$L = \text{Card}\ \theta(E). \tag{8}$$

If we think of schedules as clocks, then the latency is the total number of clock ticks, a very natural result. This definition of the latency has the advantage of being insensitive to multiplication by a positive factor.

2.2 Existence theorems

A *static control program* [2] is built from simple statements by iteration and sequence. The flow of control in such a program may be described by its sequencing predicate, $\prec$. If $(R, x) \prec (S, y)$, then operation (R, x) is executed before (S, y). I have shown elsewhere [2] that the sequencing predicate has a simple expression:

$$(R, x) \prec (S, y) \equiv x[1..N_{RS}] \ll y[1..N_{RS}] \vee (x[1..N_{RS}] = y[1..N_{RS}] \wedge T_{RS}), \tag{9}$$

where N_{RS} is the number of loops surrounding both R and S, and where T_{RS} is true iff R precedes S in the program text.

If a GDG comes from a static control program, then its dependence relation is coarser than $\prec$. Hence any schedule which satisfies:

$$u \prec v \Rightarrow \theta(u) \ll \theta(v)$$

will be causal in the sense of (7). Such a schedule always exists:

Theorem 2 *Any static control program has a multidimensional affine schedule.*

Associate to each statement R an integer vector π_R of dimension $N_{RR}+1$, the placement vector of R, with the two properties:

$$\begin{aligned} \pi_R[1..N_{RS}] &= \pi_S[1..N_{RS}], & (10)\\ T_{RS} &\equiv \pi_R[N_{RS}+1] < \pi_S[N_{RS}+1]. & (11) \end{aligned}$$

Such a vector always exists. One may, for instance, start from a Dewey Decimal Numbering of the program syntax tree. The numbers on a path from the root to statement R constitute a placement vector for R. Consider the following schedule:

$$\theta_R(x) = \pi_R \parallel x, \tag{12}$$

where $\parallel$ is the *interleave* operator. The interleave of two vectors u and v is defined by:

$$(u \parallel v)[2i-1] = u[i],$$

$$(u \parallel v)[2i] = v[i].$$

This operation may be applied to vectors of unequal dimension by extending the shorter one with zero's.

Proving that (12) reproduce the sequencing predicate is easy. In fact, there are two cases. Suppose first that:

$$x[1..N_{RS}] \ll y[1..N_{RS}].$$

Let $k \leq N_{RS}$ be the smallest index such that:

$$x[k] < y[k].$$

From (10) and the definition of $\parallel$, one deduce that:

$$\theta_R(x)[1..2k-1] = \theta_S(x)[1..2k-1],$$

$$\theta_R(x)[2k] < \theta_S(x)[2k],$$

and hence that:

$$\theta_R(x) \ll \theta_S(x).$$

If, on the contrary,

$$x[1..N_{RS}] = y[1..N_{RS}],$$

then the conclusion follows by (11).

Note that schedule (12) is not the simplest one: by using an argument similar to theorem 1, one may reduce its dimension from $2N_{SS}+1$ to $N_{SS}+1$.

The converse of the above theorem is also true:

$$\theta_R(x) \ll \theta_S(y) \Rightarrow (R,x) \prec (S,y).$$

The proof is left to the reader. ∎

From the schedule which has been constructed in the above theorem, one may deduce a piecewise polynomial schedule by theorem 1. This schedule will have a mean degree of parallelism of 1. In the special case where all domains are parallelepipeds with numerically given bounds, the schedule obtained in this way is linear[3]. Finally, in view of the following trivial

Lemma 3 *If two GDG $\mathcal{G}_1 = \langle E_1, \Gamma_1\rangle$ and $\mathcal{G}_2 = \langle E_2, \Gamma_2\rangle$ are such that $E_1 \subseteq E_2$ and $\Gamma_1 \subseteq \Gamma_2$, then the restriction of a causal schedule for $\mathcal{G}_2$ to E_1 is a causal schedule for $\mathcal{G}_1$.*

and the fact that any program instance may be embedded in a program with parallelepipeds as iteration domains, one gets Dowling's theorem [3]:

Theorem 4 *Any instance of a static control program has a linear schedule.*

The problem with the above result is that the schedule whose existence is thus proved has no parallelism. One may interpret Dowling's theorem as a guarantee that various linear programs that we are about to build always have a feasible solution. It still remains to find the optimum.

3 The basic algorithm

The following discussion will be addressed to the case of DFG: other cases may be handled with just an increase in the complexity of notations. Let θ_S be the unknown d-dimensional schedule. We will suppose that all components of θ_S are positive in $\mathcal{D}_S$; this is not a restriction in the most interesting case where the $\mathcal{D}_S$ are bounded.

The delay associated with edge e will be a vector:

$$\Delta_e(y) = \theta_{\delta(e)}(y) - \theta_{\sigma(e)}(h_e(y)) \tag{13}$$

whose components will be affine forms in the iteration vector and the structure parameters. The schedule must satisfy the following constraints:

$$y \in \mathcal{D}_S \Rightarrow \theta_S(y) \geq 0, \tag{14}$$

[3] Compare this statement to the familiar device which is used to linearize multidimensional arrays whose dimensions are numerical constants.

$$y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \gg 0. \tag{15}$$

The last inequality imply:

$$y \in \mathcal{P}_e \Rightarrow \Delta_e(y)[1] \geq 0. \tag{16}$$

The first components of (14) and (16) may be subjected to the Farkas algorithm. The only unknown is the first component of schedule θ, which will be an affine combination[4]:

$$\theta_S(y)[1] = \mu_{e1}^0 + \sum_{k=1}^{p_S} \mu_{e1}^k (a_s^k . \begin{pmatrix} y \\ n \end{pmatrix} + b_S^k), \tag{17}$$

where the μ_{e1}^k will have to satisfy a set of linear constraints:

$$G \begin{pmatrix} \mu_1 \\ \lambda_1 \end{pmatrix} \geq \mathbf{d}, \tag{18}$$

the λ_1 being other Farkas multipliers.

As we have seen above, it may or may not be possible to select in this set a solution such that:

$$y \in \mathcal{P}_e \Rightarrow \Delta_e(y)[1] > 0.$$

If such a solution is found, the problem is solved and the original DFG has a one-dimensional schedule. If not, suppose that we select in some way a particular solution of (18). Let $\mathcal{P}_{e1}$ be the set:

$$\mathcal{P}_{e1} = \{y \mid \Delta_e(y)[1] = 0\},$$

which may or may not be empty.

The next component of the schedule should satisfy:

$$y \in \mathcal{P}_{e1} \Rightarrow \Delta_e(y)[2] \geq 0, \tag{19}$$

this condition being void if $\mathcal{P}_{e1} = \emptyset$.

To characterize the set $\mathcal{P}_{e1}$, recall that in the process of solving (17), $\Delta_e(y)[1]$ has been expressed as:

$$\Delta_e(y)[1] = \lambda_{e1}^0 + \sum_{k=1}^{p_e} \lambda_{e1}^k (c_e^k . \begin{pmatrix} y \\ n \end{pmatrix} + d_e^k),$$

where all terms in the sum are nonnegative. One easily sees that:

1. if $\lambda_{e1}^0 > 0$, then $\mathcal{P}_{e1} = \emptyset$;
2. if all $\lambda_{e1}^k = 0$, then $\mathcal{P}_{e1} = \mathcal{P}_e$.

[4] The notations are those of section I.3.2

3. finally, if $\lambda_{e1}^0 = 0$ and $X_{e1} = \{k \mid \lambda_{e1}^k = 0\}$, then $\mathcal{P}_{e1}$ is a *face* of $\mathcal{P}_e$:

$$\mathcal{P}_{e1} = \{y \mid c_e^k . \begin{pmatrix} y \\ n \end{pmatrix} + d_e^k \geq 0, k \in X_{e1}, c_e^k . \begin{pmatrix} y \\ n \end{pmatrix} + d_e^k = 0, k \in [1, p_f] - X_{f1}\},$$

which may or may not be empty.

Since the sets $\mathcal{P}_{e1}$ again are polyhedra, the same method will apply. The algorithm will proceed until either all $\mathcal{P}_{ed}$ are empty, in which case a d-dimensional schedule has been found, or until $\mathcal{P}_{ei} = \mathcal{P}_{e(i+1)}$ for all e, in which case no solution exists.

It is quite obvious that this algorithm is correct and complete. All solutions will be causal multidimensional schedules, and if such a schedule exists, the algorithm will find it. The price to pay is the highly non linear character of the algorithm. The result of each step acts directly on the set of constraints to be satisfied next. Hence the solution method will be basically an enumeration. The next objective is to find a simpler version by sacrificing some of the completeness of the basic algorithm.

4 A greedy algorithm

4.1 Heuristics

The basic objective of the new algorithm is the minimization of the dimension of the schedule. This is justified by the following estimation. Suppose that the original program is processing an object of characteristic "size" n. Then, most of the time, a schedule of dimension d will have a latency $O(n^d)$.

> This will not always be true. An obvious counter-exemple is schedule (12). Let $N = \max_S N_{SS}$. Schedule (12) is of dimension $2N+1$, while its latency is probably $O(n^N)$. However, the relation will be true often enough that it can be used as a heuristics principle.

Let us say that edge e of the DFG is satisfied by a given schedule if the corresponding delay is strictly positive everywhere in $\mathcal{P}_e$. The new algorithm will be greedy in so far as it will try to satisfy as much edges as possible at each stage of the solution. Namely, suppose that at a given stage in the algorithm, U is the set of unsatisfied edges. Our aim is the determination of the next component of the schedule in such a way that as many edges of U as possible are satisfied. Since (6) is homogeneous and since the latency as given by (8) is insensitive to multiplication by a positive constant, if $e \in U$ is satisfied, then one may suppose that:

$$\Delta_e(y) \geq 1.$$

This suggest the introduction of a set of new variables x_e, $e \in U$ and the resolution of a new problem:

$$\begin{aligned} \sigma &= \max \sum_{e \in U} x_e, \\ & 0 \leq x_e \leq 1, \\ & y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \geq x_e. \end{aligned} \tag{20}$$

The resolution process will start by applying the Farkas algorithm to the last constraint. The result will be a linear program with objective function σ, the unknowns being the x_e and the Farkas multipliers.

Lemma 5 *The solution of (20) is such that all x_e are either 0 or 1.*

Let $X = \min\{x_e \mid x_e > 0\}$. Suppose $X < 1$. For all S, set $\theta'_S = \theta_S / X$. Since the delays are linear functions of the schedules, the new delay Δ' satisfies:

$$y \in \mathcal{P}_e \Rightarrow \Delta'_e(y) \geq x_e / X.$$

If $x_e = 0$, set $x'_e = 0$. If not, since $x_e/X \geq 1$, set $x'_e = 1$. The pair θ'_S, x'_e will be a solution of (20) and σ will have increased, a contradiction. ∎

Lemma 6 *The solution of (20) is unique.*

Suppose, a contrario, that we are given two distinct solutions θ^1_S, x^1_e and θ^2_S, x^2_e, obviously with the same value σ of the objective function. The pair $\theta'_S = \frac{\theta^1_S + \theta^2_S}{2}, x'_e = \frac{x^1_e + x^2_e}{2}$ is also a solution of (20). The x'_e may be adjusted to be 0 or 1 by the same process as in lemma 5. The set of satisfied edges of the new schedule will be the union of the corresponding sets of the initial schedules. If those are distinct, this will lead to an increase of σ, a contradiction. ∎

As a consequence of lemma 5, the value of σ will be the number of newly satisfied edges. If $\sigma = 0$ the algorithm fails. If $\sigma = \text{Card}\, U$, the algorithm succeeds. For all other values, we start again with a new:

$$U' = \{e \in U \mid x_e = 0\}.$$

In case of success, that the resulting multidimensional schedule is a solution of (6) is obvious. If the source DFG has a one dimensional schedule, the algorithm will terminate in one step. However, the greedy algorithm is not a variant of the basic algorithm. In fact, the greedy algorithm proceeds as if they were only two possibilities, case 1 and 2 above; case 3 is lumped with case 2. Nevertheless, theorem 2 is still true for the greedy algorithm.

4.2 Termination proof

The proof makes use of a few definitions and observations on the DFG's of static control programs. Firstly, G being a directed graph, the relation between vertices u and v: "$u = v$ or there is a cycle of G which goes through u and v" is an equivalence relation. The corresponding equivalence classes are the *strongly connected components* (scc) of G, and the quotient graph of G is acyclic.

To each edge e of G let us associate:

- an integer $N_e = N_{\sigma(e)\delta(e)}$, the number of loops which contain both the source and destination of e;
- a boolean $T_e = T_{\sigma(e)\delta(e)}$ which is true iff e "goes forward" in the program text.

Examination of the procedure which is used for computing the DFG [2] shows that to each edge e may be associated an integer d_e – the depth of e – such that:

1. if $d_e < N_e$, then:

$$\begin{aligned} h_e(y)[1..d_e] &= y[1..d_e], \\ h_e(y)[d_e+1] &< y[d_e+1]. \end{aligned}$$

2. if $d_e = N_e$, then T_e is true and:

$$h_e(y)[1..d_e] = y[1..d_e],$$

and there is no other possibility.

The next observation is that in a structured language, two loop bodies are either disjoint, or one of them includes the other. As a particular case, if two loops at the same level of nesting have one instruction in common, they are the same loop. Another consequence is that if A and B are the bodies of two distinct loops at the same level, and if there exists instructions $R \in A$, $S \in B$ such that T_{RS} is true, then all instructions of A are before any instruction of B in the program text. Hence, the notion of textual order may be extended to loop bodies at the same nesting level. The fact that all instructions of A occurs before all instructions of B in the program text will be recorded by stipulating that the boolean T_{AB} is true. As a particular consequence, it is impossible to find a cycle in the loop order: there is no $B_i, i = 0, n$ such that $T_{B_{i-1}B_i}, i = 0, \ldots, n$ and $T_{B_nB_0}$ are all true.

There is a strong connection between cycles in the DFG and loops in the source program:

Lemma 7 *Let G be a set of instructions, and let G_d be the subgraph of the DFG whose set of vertices is G and whose edges are at depth d or higher. To any cycle C of G_d one may associate $d+1$ nested loops in the source program whose bodies include all instructions of C.*

Let $R_0, R_1, \ldots, R_n = R_0$ be the instructions in C, and let e_i be the edge from R_{i-1} to R_i. Obviously, $N_{e_i} \geq d$. Hence, there are at least d loops whose bodies contain R_{i-1} and R_i, and also d loops which contain R_i and R_{i+1}. From an observation above, these d loops are identical since they share an instruction. Continuing in this way around C, we conclude that there are at least d loops which contain all instructions in C.

Suppose that at level $d+1$ these instructions belong to several distinct loops $B_1, \ldots B_m$. Suppose that these loops are enumerated in textual order. Since C is a cycle, there necessarily is a "backward edge" e from $R \in B_i$ to $S \in B_j, i > j$. By definition, $d_e \leq d$, but since B_i and B_j are distinct loops at nesting level $d+1$, $d_e \leq N_e = d$. Hence $d_e = N_e$, and we are in case 2 of the definition of the depth of e. As a consequence, T_e is true, which contradict the fact that e is a backward edge. ∎

Theorem 8 *Let U be an arbitrary subset of the edges in the DFG of a static control program. Then solving problem (20) for U always satisfies at least one edge of U.*

Let G be the set of instructions incident to U, and let d be the minimum depth of edges in U. Let us form the graph G_d, which contains all edges of U. Let $H_1, \ldots, H_n$ be the scc's of G_d, and let Γ be its quotient graph. There are two cases:

- There exists in U at least one edge e which connects two distinct scc's. Affix to each scc H_i an integer Λ_i in such a way that:

  $$\langle H_i, H_j \rangle \in \Gamma \Rightarrow \Lambda_i < \Lambda_j.$$

  Such integers always exist since Γ is acyclic, and may be easily found by the topological sort algorithm. Now set:

  $$\forall R \in H_i : \theta_R(x) = \Lambda_i.$$

  It is clear that for all edges which lie inside one scc, the delay will be 0, while if there is an edge from R in H_i to S in H_j, then $\langle H_i, H_j \rangle \in \Gamma$ and:

  $$\theta_S(y) - \theta_R(x) = \Lambda_j - \Lambda_i > 0.$$

- All edges are internal to some scc. Let us take:

  $$\theta_R(x) = x[d+1].$$

  This is a legitimate choice, since in this case all instructions belong to at least $d+1$ loops. Let e be an edge of U whose source and destination are R and S. By lemma 7, $N_e \geq d+1$. If $d_e > d$, then the associated delay is zero. if $d_e = d$, we are in the first case of the definition of d_e:

  $$\Delta = x[d_e+1] - h_e(x)[d_e+1] > 0,$$

  and e is satisfied. Since, by the definition of d, there is at least an edge such that $d_e = d$, it follows that there is at least one satisfied edge.

∎

It is now an easy corollary to theorem 8 that for a static control program, the greedy algorithm always succeeds.

4.3 The dual version of the greedy algorithm

In the first part of this paper, we have found that among all schedules which satisfy the causality constraint, we may select the best one by dualization. In the present context, this would mean a two-step process: determine first the set of satisfied edges, as above, then find the best concave schedule. Our aim now is to find a way to solve this problem with just one linear program.

In order to simplify the notations, let μ be a vector whose elements are the unknowns in the problem, i.e. the x_e, the μ_S^k and some of the λ_e^k. Let u be a vector with ones in positions corresponding to the x_e:

$$\sum_e x_e = u.\mu.$$

Similarly, let $\phi_S(x)$ be a vector such that the schedule for instruction S is given by:

$$\theta_S(x) = \phi_S(x).\mu.$$

Lastly, let

$$G\mu \geq d$$

be the set of constraints which are deduced from the causality condition by the Farkas algorithm, including the constraints $x_e \leq 1$. The first problem is simply a restatement of (20):

$$\begin{aligned} \sigma &= \max u.\mu, \\ &\mu \geq 0, \\ &G\mu \geq d. \end{aligned} \tag{21}$$

Each schedule is given by the solution of another problem:

$$\begin{aligned} \theta_S(x) &= \min \phi_S(x).\mu, \\ & \mu \geq 0, \\ & u.\mu = \sigma, \\ & G\mu \geq d. \end{aligned} \tag{22}$$

Since the solution of (21) is unique, the effect of the second constraint in (22) is to pin the values of the x_e to their optimum values. Let us set up a third problem, in which m is a new positive parameter:

$$\begin{aligned} \tau &= \max m u.\mu - \phi_S(x).\mu, \\ & \mu \geq 0, \\ & G\mu \geq d. \end{aligned} \tag{23}$$

Intuitively, we will be interested in what happens when m grows very large. Let Y_1 (resp. Y_2) be the solution of (21) (resp. (23)). Both vectors are feasible points for the other problem. Hence:

$$u.Y_2 \leq u.Y_1, \tag{24}$$
$$mu.Y_1 - \phi_S(x).Y_1 \leq mu.Y_2 - \phi_S(x).Y_2. \tag{25}$$

As a consequence:

$$m(u.Y_1 - u.Y_2) \leq \phi_S(x).Y_1 - \phi_S(x).Y_2 \leq \phi_S(x).Y_1.$$

The last bound results from the fact that both $\phi_S(x)$ and Y_2 have positive components. Since the last bound is independent of m, this inequality will be possible for a large enough value of m iff $u.Y_2 = u.Y_1 = \sigma$. For a large enough m, solving (23) will thus be equivalent to solving (22).

The solution of (23) is found by applying the PIP algorithm to its dual:

$$\begin{aligned} \tau &= \min \nu.d, \\ & \nu \geq 0, \\ & \nu G \leq mu - \phi_S(x). \end{aligned} \tag{26}$$

The solution for τ will be a quast on the parameters m and x. To obtain the limit when $m \to \infty$, one simply has to take the true branch on all tests in which m occurs with a positive coefficient and conversely. As has been explained elsewhere [2], these simplifications can be done "on the fly", so as to keep the extra work to a minimum. The coefficient of m in the solution will give the value of σ, and the independent term will be $-\theta_S$. The values of the x_e may be recovered at no extra cost by using *complementary slackness* [4, Chap. 5]: if some component of ν is nonzero, the solution of the primal problem satisfies the corresponding constraint as an equality. Now, among the constraints of (23) are

$$x_e \leq 1.$$

If the corresponding dual solution is nonzero, $x_e = 1$ and edge e is satisfied. By comparing the number of satisfied edges with σ, it is a simple matter to check that all satisfied edges are found in this way.

5 Decomposition in strongly connected components

I wish now to investigate the application of the "Divide and Conquer" design rule to the above algorithms. If one consider the causality rule as a set of constraints on causal schedules, one sees that the GDG (or the DFG), when shorn of its ancillary information – like domains and dependence polyhedra – gives a pictorial representation of the relationships between the unknowns of the problem: there is an edge from R to S iff θ_R occurs in a constraint for θ_S. Suppose that the set of vertices of the GDG may be partitionned in two subsets V_1 and V_2 in such a way that there are edges from V_1 to V_2 but no edge from V_2 to V_1. It seems plausible that the scheduling problem for V_1 may be solved first, and then the results be interpolated in the constraints for V_2. This process will be called step-by-step scheduling.

5.1 Basic results

To be precise, let us consider the following sets:

- $\mathcal{P}$, the set of causal schedules for the whole GDG,
- $\mathcal{P}_1$, the set of causal schedules for the vertices of V_1,
- $\mathcal{P}_2$, the set of causal schedules for the vertices of V_2, the constraints associated to the edges from V_2 to V_1 being ignored.

Lemma 9 *The restriction of a schedule of $\mathcal{P}$ to the vertices of V_1, (resp. V_2) is a schedule of $\mathcal{P}_1$, (resp. $\mathcal{P}_2$).*

Obvious. ∎

Lemma 10 *If the domains of the given GDG are finite sets, then from a schedule t^1 of $\mathcal{P}_1$ and a schedule t^2 of $\mathcal{P}_2$ one may build a schedule of $\mathcal{P}$.*

According to lemma I.8, there exists an affine form $L(n)$ such that:

$$\forall R \in V_1,\, t^1_R(x) \leq L(n). \tag{27}$$

Consider the schedule:

$$\begin{aligned} t_R(x) &= t^1_R(x),\, R \in V_1, \\ t_S(x) &= t^2_S(x) + L(n),\, S \in V_2. \end{aligned}$$

t satifies edges internal to V_1 and V_2 by definition of t^1 and t^2. Edges from V_2 to V_1 are satisfied by (27). ■

From these results one deduces that, in the case of finite domains, $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) is the restriction of $\mathcal{P}$ to V_1 (resp. V_2, up to a translation in that case). Suppose now that $\mathcal{P}$ is not empty, which implies that neither $\mathcal{P}_1$ nor $\mathcal{P}_2$ are. Let us consider problem (21). Suppose that the optimum solution in V_1, t^1 is different from the optimum in the whole of V, t. t^1 may be extended to a solution in V, then superposed to t, as in the proof of lemma 6, thus contradicting the unicity of the solution. As a consequence we see that the solution of (21) may be undertaken step-by-step.

A similar result holds for (22), since the objective function is purely local. As a consequence, (23) and (26) can be solved step-by-step.

This results holds neither for minimum latency schedules nor for bounded delay schedules. For minimum latency schedules, a counter-exemple is given by Darte et. al. [5, Fig. 7]. For bounded delay schedules, observe that the program in Fig I.2 has no such schedule, while each of its instructions has one.

5.2 A Recursive Scheduling Algorithm

These results may be generalized in the following way. Let $\{H_1, \ldots, H_n\}$ be the strongly connected components of the GDG, their enumeration being compatible with the reduced graph. A simple induction will show that the scheduling problem may be solved step-by-step, starting with H_1, each solution t^i being used if necessary in setting up the constraints for $t^j, j > i$. Since t^i may be piecewise affine, one may have to apply Farkas Lemma independently to each piece, a straightforward extension of algorithms in part I.

For each H_i, one will attempt the solution of a program like (26). There will be one set of constraints per edge incident to vertices in H_i. These edges may be classified into:

- *internal edges*, whose source lies in H_i,
- *external edges*, whose source lies in $H_k, k < i$.

It is easy to see that external edges will always be satisfied in the solution – if not, simply add 1 to the schedule. If there are unsatisfied internal edges, one must start again for the restriction of the GDG to unsatisfied edges. This subgraph which is not necessarily strongly connected, must be decomposed again. The outcome of these observation is the following algorithm:

- **Schedule**(U, p):
- U is a set of edges in the GDG and p is an integer. Initially, $p = 1$ and U is the set of all edges in the GDG.

  1. Compute the strongly connected components of U, $\{H_1, \dots, H_n\}$, ranking them according to the reduced graph of U.
  2. For each $i = 1, \dots, n$, solve (26).
     (a) If the solution is such that $\sigma = 0$, the algorithm fails. This never happens if the GDG comes from a sequential program.
     (b) If not, the schedules obtained at step 2 are the components of index p of the multidimensional schedule.
     (c) Build the set U' of unsatisfied edges, and, if $U' \neq \emptyset$, call recursively **Schedule**$(U', p + 1)$.

The similarity of this algorithm with the **codegen** algorithm of Allen and Kennedy [6] is striking. In fact, one may conjecture that **codegen** is a simplified version of **Schedule** in which program (26), instead of being solved exactly, is solved approximately along the lines of theorem 8.

6 Experimental results

I wish now to explore the scheduling algorithms of this paper from a more practical point of view. Part I of the present paper presents two basic algorithms, the bounded delay scheduler of section I.3.3.2 and the dual algorithm of section I.3.3.3. In part II, I have shown how to extend the dual algorithm to the multidimensional case. The bounded delay version may be similarly extended, notwithstanding the fact that the definition of a delay for multidimensional time is somewhat unclear. Lastly, I have presented a partitioning technique, which is guaranteed to give the same result as a global solution for the dual algorithm but not for the bounded delay case. All these algorithms have been implemented in such a way as to allow complete freedom in their combination. This software has been applied to a collection of kernels from numerical analysis and signal processing. The following is an assessment of the results of these experiments.

6.1 Implementation

The present software computes schedules for affine Dataflow Graphs only; extension to the case of Generalized Dependence graphs is contemplated. The architecture of the software closely follows algorithm **Schedule** above. The first step is the acquisition of the DFG. Optionally, one may ask for the elimination of redundant inequalities in the definition of domains. **Schedule** is then applied to the whole program. The second option may be used to inhibit the scc decomposition: the whole graph is considered to be strongly connected whatever its shape. Lastly, when solving a scheduling problem, one may ask either for the best concave solution or for the bounded delay solution. In the last case, the delay condition is imposed only to internal edges of each scc.

Application of Farkas Lemma implies the handling of sparse linear and bilinear forms; this is best implemented in a symbolic programming language like Lisp. The code has been structured in two layers: the outermost one implements the scheduling algorithm, the inner layer being a rudimentary algebraic calculator. The result is a parametric integer programming problem, which is sent to the PIP software. PIP is written in C and has been described elsewhere [7, 8].

6.2 Some exemples

The program in fig. 2 is a Fortran implementation of the familiar Gaussian elimination algorithm. The only unusual feature is the back substitution – the `ip` loop – in which subscripts have been rewritten in order to avoid negative increments.

Table 2 is a list of the instructions in the program, with a description of each iteration domain. Table 3 is a description of the associated Dataflow Graph. Edges are numbered arbitrarily by the software.

An attempt to build a one-dimensional schedule for this program does not succeed. In fact, one may observe that the basic structure of the back-substitution part is the same as that of the program of fig. 1. Execution of algorithm **Schedule**, however, is successful. The DFG may be divided into three strongly connected components: $\{3\}$, $\{1, 2\}$ and $\{4, 5\}$. Instruction $\{3\}$ has no predecessor, hence its schedule is zero, which means that it can be executed outright at the beginning of the program. Instructions 1 and 2 have an affine schedule, which is given in table 2. When scheduling instructions 4 and 5, one find that edge 109 cannot be satisfied. Algorithm **Schedule** is called recursively. The unsatisfied

```
    program gosse
    real a(100,100), x(100), s
    integer i, j, k, n
    real f
    do i=1, n-1
      do j=i+1, n
1       f=a(j,i)/a(i,i)
        do k=i+1,n+1
2         a(j,k)=a(j,k)-f*a(i,k)
        end do
      end do
    end do
    do ip=1,n
3     s = 0.
      do j=n-ip+2, n
4       s = s + a(n-ip+1, j)*x(j)
      end do
5     x(n-ip+1) = (a(n-ip+1, n+1) - s)/a(n-ip+1,n-ip+1)
    end do
    end
```

Figure 2: Gaussian elimination

Instruction	Loop counters	Schedule	Domain
1	i, j	$2i-2$	$n-1-i \geq 0$, $i-1 \geq 0$, $n-j \geq 0$, $j-1-i \geq 0$
2	i, j, k	$2i-1$	$n-1-i \geq 0$, $i-1 \geq 0$, $n-j \geq 0$, $j-1-i \geq 0$, $1+n-k \geq 0$, $k-1-i \geq 0$
3	ip	0	$n-ip \geq 0$, $ip-1 \geq 0$
4	ip, j	$\begin{pmatrix} 2ip+2n-4 \\ ip+j-2-n \end{pmatrix}$	$n-ip \geq 0$, $ip-1 \geq 0$, $n-j \geq 0$, $ip+j-2-n \geq 0$
5	ip	$2ip+2n-3$	$n-ip \geq 0$, $ip-1 \geq 0$

Table 2: The iteration domains of the `gosse` program

Edge	Source	Destination	Delay	Predicate
101	$\langle 1, i, j \rangle$	$\langle 2, i, j, k \rangle$	1	
102	$\langle 2, i-1, j, i \rangle$	$\langle 1, i, j \rangle$	1	$i-2 \geq 0$
103	$\langle 2, i-1, i, i \rangle$	$\langle 1, i, j \rangle$	1	$i-2 \geq 0$
104	$\langle 2, i-1, j, k \rangle$	$\langle 2, i, j, k \rangle$	2	$i-2 \geq 0$
105	$\langle 2, i-1, i, k \rangle$	$\langle 2, i, j, k \rangle$	2	$i-2 \geq 0$
106	$\langle 2, n-ip, 1+n-ip, j \rangle$	$\langle 4, ip, j \rangle$	$4ip-3$	$n-1-ip \geq 0$
107	$\langle 2, n-ip, 1+n-ip, 1+n \rangle$	$\langle 5, ip \rangle$	$4ip-2$	$n-1-ip \geq 0$
108	$\langle 2, n-ip, 1+n-ip, 1+n-ip \rangle$	$\langle 5, ip \rangle$	$4ip-2$	$n-1-ip \geq 0$
109	$\langle 4, ip, j-1 \rangle$	$\langle 4, ip, j \rangle$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$ip+j-3-n \geq 0$
110	$\langle 3, ip \rangle$	$\langle 4, ip, j \rangle$	$2ip+2n-4$	$2+n-ip-j \geq 0$
111	$\langle 4, ip, n \rangle$	$\langle 5, ip \rangle$	1	$ip-2 \geq 0$
112	$\langle 3, ip \rangle$	$\langle 5, ip \rangle$	$2ip+2n-3$	$1-ip \geq 0$
113	$\langle 5, 1+n-j \rangle$	$\langle 4, ip, j \rangle$	$2ip+2j-3-2n$	

Table 3: The DFG of the `gosse` program

	Setup	PIP	Total
scc	4.2	1.5	5.7
no scc	11	15	26

Table 4: The time taken for computing the schedules in Fig. 2

edge is a loop from instruction 4 to itself. The recursive call build a second component for the schedule of 4. The results of algorithm **Schedule** is displayed in table 2. An important information is the delay of each edge: it may be found in table 3; the reader may care to verify that all delays are lexicographically positive. In the case of edge 113, for instance, the property

$$2ip + 2j - 3 - 2n > 0$$

is a consequence of the last inequality in the domain of instruction 4:

$$ip + j - 2 - n \geq 0.$$

Computing these schedules may also be done without resorting to the scc decomposition of section 5. The results are exactly the same; however, the running time is quite different, as shown by table 4.

Times were measured on a Sparc station whith the standard operating system tools and are given in seconds. The "setup" column represent the time it takes to construct the linear program (26) (this is the Lisp part of the software), while the "PIP" column represent the time for its solution by the parametric programming code, which is written in C. The scc decomposition is responsible for a fourfold reduction in the running time, which is mostly assignable to the reduction in the linear program size.

It is clear from table 3, that the resulting schedule has unbounded delays – the largest delay is of the order of $4n$. An attempt to build a bounded delay schedule fails.

The fact that one of the schedules in table 2 is two dimensional means that the latency is $O(n^2)$. This may be understood as a measure of the running time on a computer with at least n^2 processors. A more realistic interpretation is that one will need $O(n^2)$ synchronization operations if no parallelism is to be lost. This is an unsatisfactory situation, which can be remedied by a well known device. Observe that lines 3 and 4 of the above program compute a sum:

$$s = \sum_{j=n-ip+2}^{n} a_{n-ip+1,j} x_j .$$

Now, if one forget about rounding errors, this sum may be computed in any order,

```
      program gosser
      real a(100,100), x(100), s
      integer i, j, k, n
      real f
      do i=1, n-1
        do j=i+1, n
1         f=a(j,i)/a(i,i)
          do k=i+1,n+1
2           a(j,k)=a(j,k)-f*a(i,k)
          end do
        end do
      end do
      do ip = 1, n
3       s = 0.
        do jp = 1, ip -1
4         s = s + a(n-ip+1, n-jp+1)*x(n-jp+1)
        end do
5       x(n-ip+1) = (a(n-ip+1, n+1) - s)/a(n-ip+1, n-ip+1)
      end do

      end
```

Figure 3: Another version of a Gaussian elimination program

for instance:

$$s = \sum_{j=n}^{n-ip+2} a_{n-ip+1,j} x_j = \sum_{j'=1}^{ip-1} a_{n-ip+1,n-j'+1} x_{n-j'+1}.$$

After some rearrangements, one gets the program in fig. 3, which has a one dimensional schedule: see table 5.

The new schedule has an $O(n)$ latency, but its delays are still not bounded. Trying to compute a bounded delay schedule while still using the scc decomposition gives the same result as table 5. This is a consequence of the fact that we do not enforce the bounded delay condition on "external" edges. With the scc decomposition off, a two dimensional schedule results (see table 6). One sees that the first component of each schedule is either 0 or 1, thus reconstructing the scc decomposition, and that the other component is the same, up to a translation, as the solution in table 5. The conclusion is that program `gosser` in fact has no bounded delay schedule. However, it may be decomposed in two bounded delay phases, with an unbounded delay between them. The elapsed times for solving these problems are given in table 7.

Instruction	Loop counters	Schedule
1	i, j	$2i - 2$
2	i, j, k	$2i - 1$
3	ip	0
4	ip, jp	$2jp + 2n - 2$
5	ip	$2ip + 2n - 3$

Table 5: The schedules for program `gosser`

Instruction	Loop counters	Schedule
1	i, j	$\begin{pmatrix} 0 \\ 2i - 2 \end{pmatrix}$
2	i, j, k	$\begin{pmatrix} 0 \\ 2i - 1 \end{pmatrix}$
3	ip	0
4	ip, jp	$\begin{pmatrix} 1 \\ 2jp - 1 \end{pmatrix}$
5	ip	$\begin{pmatrix} 1 \\ 2ip - 2 \end{pmatrix}$

Table 6: Scheduling `gosser`, no scc decomposition

	Setup	PIP	Total
best schedule, scc	4	1.5	5.5
best schedule, no scc	10.6	18	28.6
bounded delay, scc	4.6	1	5.6
bounded delay, no scc	13.2	10	23.2

Table 7: The time taken for scheduling `gosser`

```
      program s211
c
c     statement reordering
c     statement reordering allows vectorization
c
      integer n
      real a(100),b(100),c(100)
      do 270 i = 2,n-1
1        a(i) = b(i-1) + c(i)
2        b(i) = b(i+1) - 2.
  270 continue
c     return
      end
```

Figure 4: An exemple from the Argonne test suite

As a last exemple, consider the programm in fig. 4 – an extract from the Argonne test suite. The bounded delay schedule for this program is:

$$\begin{aligned} t_1(i) &= 1, \\ t_2(i) &= 0, \end{aligned}$$

while the best concave schedule is:

$$\begin{aligned} \tau_1(i) &= \textbf{if } i-4 \geq 0 \textbf{ then } 1 \textbf{ else } i-2, \\ \tau_2(i) &= 0. \end{aligned}$$

The bounded delay schedule is very simple. It indicates that both instructions can be executed as separate parallel loops, provided that their order be inverted. Note that this result is only valid due to the implicit renaming of array `b` which is implied by the DFG calculation process. The best schedule is slightly more complicated: the dual algorithm has noticed that the flow dependence from `b(i)` in instruction 2 to `b(i-1)` in the next iteration of instruction 1 does not exists for $i = 2$. Hence, the first iteration of instruction 1 can be executed at time 0 rather than 1.

On the basis of these observations, we may conclude that the computation of best concave schedules is the safest solution for a restructuring compiler. Such schedules always exist and are unique; their only drawback is that they may be slightly more complicated than bounded delay schedules, and that the added complication may no be worthwhile in term of performance – see the last exemple. One may contemplate a two step approach, in which one try first for a bounded delay schedule and then, in case of failure, for a best schedule. Evaluation of such tactics must wait for much more practical experience.

6.3 The limits of the algorithm

In its present implementation, there are two limiting factors to the applicability of the algorithm – beyond the standard restriction to static control programs with linear subscripts. Both are related to the "size" of problem (26). One limitation is in term of the number of constraints and unknowns. The largest program that was ever scheduled had 24 assignment statements and a maximum nesting level of 4. This led to the solution of several linear programs, the largest of which had 980 constraints and 581 unknowns. The elapsed time was about two hours on a low end workstation. The second limitation is in term of an upper bound b on the modulus of the elements of matrix G in (26). Finding the exact solution of this problem will entails the handling of integers of the order of b^m where m is the number of unknowns. Since the present implementation of PIP uses ordinary 32 bits arithmetic, this will soon induces overflows unless b is 1 or 2 at most. As an irritating consequence, while the scheduler has no difficulty with programs like `gosser` or `s211`, it fails if the loops upper bounds (e.g., `n`) are replaced by numerical constants.

There are several ways out of these difficulties. One of them is to detect large loop bounds and replace them by new parameters, then to plug back the numerical values in the resulting schedule.

The scc decomposition has the effect of partitioning the program into phases no larger than the loop nests in the original program. This has a limiting effect on the problem size. However, there will be programs which surpass the possibilities of our implementation. A possible solution in that case is to implement an approximate scheduler along the lines of theorem 8. The approximate schedule will satisfy some edges of the DFG. Algorithm **Schedule** may then be called for solving a deflated problem. These techniques and others will be the subject of future studies.

7 Conclusion

This concludes the discussion of a set of general methods for computing linear and piecewise linear, multidimensional schedules for static control programs and systems of recurrence equations. Let us emphasize the fact that the method is not limited to the case of uniform dependences, while being able to exploit such to simplify the solution process.

The reasoning which underlies the method is in two parts. Firstly, one has to characterize the set of acceptable schedules, in the form of a polyhedron in the space of the coefficients of all positive linear functions. This may be done either by the vertex method or with the help of Farkas lemma. It is probable that both these methods give logically equivalent results; it would be very interesting to do a detailed performance comparison.

The second step is the selection of a "best" – or at least, of a "good" – schedule from the above set. Here the dual method shows itself as superior to the minimum latency method and the bounded delay method, both from the point of view of determinacy – the best concave schedule is unique while there are many minimum latency and minimum delay schedules – and from the fact that it lends itself to a divide and conquer strategy, which is not true for the other methods.

In theory at least, the use of schedules for the construction of parallel program is far superior to the usual transformative approach, in so far as the equivalent parallel program is found without any state space search. The main drawback of the present proposal is its complexity, which forbid its use for anything but the smallest kernels. What is needed here is a combination of problem reduction rules, approximation techniques and schedule combination methods for tackling, at least in an approximate way, lifesize problems.

8 Acknowledgments

This work has been partially supported by the french Ministry of Defense under contract DRET 87/280. Many ideas have been evolved in discussions with Patrice Quinton, Alain Darte, and all my friends of the "Techniques de Parallélisation" operation of PRC C^3.

References

[1] Paul Feautrier. *Somme Efficient Solution to the Affine Scheduling Problem, Part I, One-dimensional Time.* Technical Report 92.28, IBP–MASI, May 1992.

[2] Paul Feautrier. Dataflow analysis of scalar and array references. *Int. Journal of Parallel Programming*, 20(1):23–53, February 1991.

[3] Michael L. Dowling. Optimal code parallelization using unimodular transformations. *Parallel Computing*, 16:157–171, 1990.

[4] A. Schrijver. *Theory of linear and integer programming.* Wiley, NewYork, 1986.

[5] Alain Darte and Yves Robert. *Affine-by-statement Scheduling of Uniform Loop Nest over Parametric Domains.* Technical Report 92-16, LIP-IMAG, April 1992.

[6] Randy Allen and Ken Kennedy. Automatic translation of fortran programs to vector form. *ACM TOPLAS*, 9(4):491–542, October 1987.

[7] Paul Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, September 1988.

[8] Paul Feautrier and Nadia Tawbi. *Résolution de Systèmes d'Inéquations Linéaires ; mode d'emploi du logiciel PIP.* Technical Report 90.2, IBP-MASI, January 1990.

```
@TECHREPORT{Feau:92b,
        AUTHOR = "Paul Feautrier",
        TITLE = "Somme Efficient Solution to the Affine Scheduling
        Problem, Part II, Multidimensional Time",
        INSTITUTION = "IBP/MASI", NUMBER = "92.78",
        YEAR = 1992, MONTH = Oct}
```