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

Paul Feautrier*
Laboratoire PRiSM,
Université de Versailles St-Quentin
45 Avenue des Etats-Unis
78035 VERSAILLES CEDEX FRANCE

February 17, 2009

Abstract

This paper extends the algorithms which were developed 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 connection 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 proposés 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

*e-mail : Paul.Feautrier@prism.uvsq.fr

```
      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

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.

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.

An example of such a program is given in Fig. 1. Application of the methods of Part I shows that this program – hereafter referred to as program 1 – has no affine schedule. It is possible to prove that its free schedule is :

$$\theta_1(i,j) = \frac{i(i+1)}{2} + j, \tag{1}$$

which has a mean degree of parallelism of 1. Hence, the original program is totally sequential[1].

However, the automatic construction of polynomials schedules seems to be beyond present day techniques. Examination of program 1 and other similar examples suggests another solution: the use of multidimensional time. This is nothing out of the way: a clock with two hands define a two dimensional time, each hand being associated with one dimension. The order on such a time is lexicographic ordering. Program 1, for instance, has the following two-dimensional schedule:

$$\theta_2(i,j) = \begin{pmatrix} i \\ j \end{pmatrix}, \tag{2}$$

[1]For definiteness, I will suppose that + in this program stands for some operator with no special algebraic properties. 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. This remark will stand for all examples in this paper. Computing schedules for operators with nontrivial algebraic properties is a largely open problem.

which will be seen later to be equivalent to schedule (1).

For usual clocks, the minor components of time are always uniformly bounded. As a consequence, their time may be linearized if necessary. This restriction is not enforced for multidimensional schedules, as the preceding example shows.

This paper is devoted to the design of algorithms for the construction of multidimensional schedules. In section 2, I will discuss the notion 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 [2].

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 The causality condition

The first step in the construction of a multidimensional schedule is to find the correct generalization of the causality condition:

$$u, v \in E,\ u\Gamma v \Rightarrow \theta(u) < \theta(v). \tag{3}$$

An obvious possibility is:

$$u, v \in E,\ u\Gamma v \Rightarrow \theta(u) \ll \theta(v). \tag{4}$$

This is in a sense sufficient, as shown by

Theorem 1 *To any d-dimensional schedule θ one may associate a one-dimensional schedule τ, namely:*

$$\tau(u) = \mathrm{Card}\ \{\theta(\mathrm{x}) \mid \theta(\mathrm{x}) \lneqq \theta(\mathrm{u}) \wedge \mathrm{x} \in \mathrm{E}\}.$$

The order relations associated to θ and τ are the same, and τ satisfies condition (3) iff θ satisfies condition (4).

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

Proof Let P_u be the set $\{\theta(x) \mid \theta(x) \not\ll \theta(u) \wedge x \in E\}$. It is easy to see that P_u depends only on $\theta(u)$, and is a strictly increasing function of $\theta(u)$:

$$\theta(u) \ll \theta(v) \Rightarrow P_u \subset P_v.$$

In fact, we have $P_u \subseteq P_v$ by transitivity, $\theta(v) \in P_v$ by definition, and supposing $\theta(v) \in P_u$ would entail a contradiction.

The order relation associated to θ is:

$$u \prec_\theta v \equiv \theta(u) \ll \theta(v).$$

Similarly:

$$u \prec_\tau v \equiv \tau(u) < \tau(v).$$

Now, obviously:

$$u \prec_\theta v \Rightarrow \theta(u) \ll \theta(v) \Rightarrow P_u \subset P_v \Rightarrow \text{Card P}_\text{u} < \text{Card P}_\text{v} \Rightarrow \tau(\text{u}) < \tau(\text{v}).$$

As a consequence, if θ is causal, $u \Gamma v$ implies $\theta(u) \ll \theta(v)$, which implies in turn $\tau(u) < \tau(v)$. Conversely, suppose $\tau(u) < \tau(v)$, which implies $\text{Card P}_\text{u} < \text{Card P}_\text{v}$. Since $\ll$ is a total order, we have either $\theta(u) \ll \theta(v)$, $\theta(u) = \theta(v)$ or $\theta(u) \gg \theta(v)$. This gives three possibilities: $P_u \subset P_v$, $P_u = P_v$ and $P_u \supset P_v$, and the first is the only one which does not lead to a contradiction.

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

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

$$L = \text{Card } \theta(\text{E}). \tag{5}$$

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 composition with a monotone increasing function, like addition of a constant or multiplication by a positive factor. The schedule which is constructed from Equ. (2) by the above method is precisely Equ. (1). In this case, θ is the identity. This implies that $\text{Card } \theta(\text{E}) = \text{Card E}$, which is another way of saying that schedule (2) has no parallelism.

2.2 Existence theorems

A *static control program* [3] is built from simple statements by bounded 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 [3] that the sequencing predicate has a simple expression:

$$(R, x) \prec (S, y) \equiv x[1..N_{RS}] \ll y[1..N_{RS}] \vee (x[1..N_{RS}] = y[1..N_{RS}] \wedge T_{RS}), \tag{6}$$

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:

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

is causal in the sense of condition (4). Such a schedule always exists:

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

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

$$\begin{aligned} \pi_R[1..N_{RS}] &= \pi_S[1..N_{RS}], & (7)\\ T_{RS} &\equiv \pi_R[N_{RS}+1] < \pi_S[N_{RS}+1]. & (8) \end{aligned}$$

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:

$$\theta(R,x) = \pi_R \parallel x, \tag{9}$$

where $\parallel$ is the *shuffle* operator. The shuffle 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 schedule (9) reproduces 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 property (7) and the definition of $\parallel$, one deduces that:

$$\theta(R,x)[1..2k-1] = \theta(S,x)[1..2k-1],$$

$$\theta(R,x)[2k] < \theta(S,x)[2k],$$

and hence that:

$$\theta(R,x) \ll \theta(S,x).$$

```
      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 i=1,n
3       s = 0.
        do j=n-i+2, n
4         s = s + a(n-i+1, j)*x(j)
        end do
5       x(n-i+1) = (a(n-i+1, n+1) - s)/a(n-i+1,n-i+1)
      end do
      end
```

Figure 2: Gaussian elimination

If, on the contrary,

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

then the conclusion follows by property (8).

Note that schedule (9) 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. ■

The program of Fig. 2 is an implementation of a Gaussian linear equation solver. Lines 1 and 2 implement the elimination phase, while lines 3 to 5 are the backward substitution phase. The subscripts in the latter phase have been modified so as to avoid the use of a negative increment in the second `do i` loop, which cannot be handled by the present version of the scheduler.

The abstract syntax tree of program 2, with Dewey Decimal Numbers, is depicted in Fig. 3.

Some examples of schedules for this program are:

$$\theta(3,i) = \langle 1, i, 0 \rangle,$$

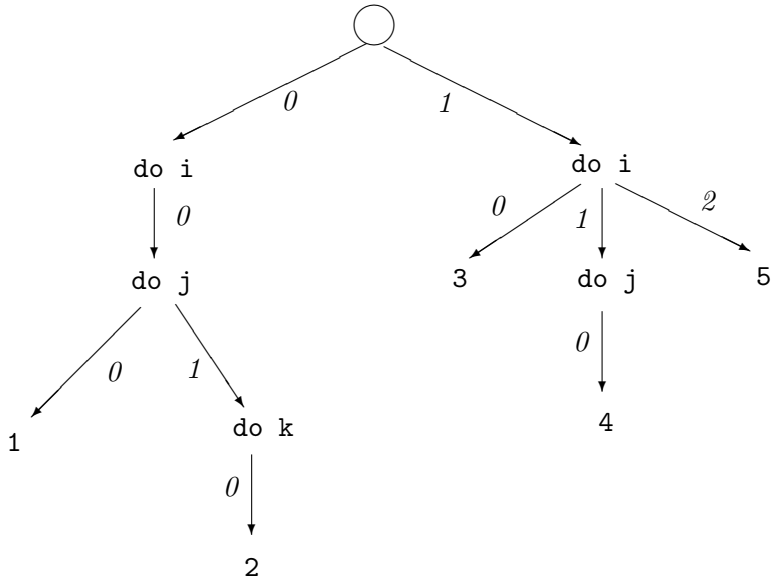

Figure 3: The abstract syntax tree of program 2

$$\theta(4, i, j) = \langle 1, i, 1, j, 0 \rangle,$$
$$\theta(5, i) = \langle 1, i, 2 \rangle.$$

The reader may care to verify that these schedules do capture the sequential execution of program 2. They may be simplified in several ways, for instance by removing useless components which arise from nodes with only one successor.

From the schedule which has been constructed in the above theorem, one may deduce a piecewise polynomial schedule by theorem 1. This schedule has 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 [4]:

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

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

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 is addressed to the case of DFG scheduling: other cases may be handled with just an increase in the complexity of notations. Let θ be the unknown d-dimensional schedule. We will suppose that all components of θ 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 is a vector:

$$\Delta_e(y) = \theta(\delta(e), y) - \theta(\sigma(e), h_e(y)) \tag{10}$$

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

$$y \in \mathcal{D}_S \Rightarrow \theta(S, y) \geq 0, \tag{11}$$

$$y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \gg 0. \tag{12}$$

The last inequality imply:

$$y \in \mathcal{P}_e \Rightarrow \Delta_e(y)[1] \geq 0. \tag{13}$$

The first components of inequalities (11) and (13) may be subjected to the Farkas algorithm. The only unknown is the first component of schedule θ, which is an affine combination[4]:

$$\theta(S, y)[1] = \mu^{(1)}_{e0} + \sum_{k=1}^{p_S} \mu^{(1)}_{ek}(a_{Sk}. \begin{pmatrix} y \\ n \end{pmatrix} + b_{Sk}), \tag{14}$$

where the $\mu^{(1)}_{ek}$ have to satisfy a set of linear constraints:

$$G \begin{pmatrix} \mu^{(1)} \\ \lambda^{(1)} \end{pmatrix} \geq d, \tag{15}$$

the $\lambda^{(1)}$ being auxiliary Farkas multipliers.

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

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

solution of problem (15). A possibility is that we partition the set of edges of the DFG into $U^{(1)}$ such that:

$$e \in U^{(1)} \Rightarrow \Delta_e(y)[1] \geq 0,$$

and its complement where we have:

$$e \notin U^{(1)} \Rightarrow \Delta_e(y)[1] > 0,$$

Since the problem is homogeneous, this condition may be rewritten:

$$e \notin U^{(1)} \Rightarrow \Delta_e(y)[1] \geq 1.$$

Such a partition is admissible if the resulting problem is found to have a solution after application of the Farkas algorithm I.3.2. Let $\mathcal{P}_e^{(1)}$ be the set:

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

which may or may not be empty.

The next component of the schedule must satisfy:

$$y \in \mathcal{P}_e^{(1)} \Rightarrow \Delta_e(y)[2] \geq 0, \tag{16}$$

this condition being void when $\mathcal{P}_e^{(1)} = \emptyset$.

To characterize the set $\mathcal{P}_e^{(1)}$, recall that in the process of solving inequality (13), $\Delta_e(y)[1]$ has been expressed as:

$$\Delta_e(y)[1] = \lambda_{e0}^{(1)} + \sum_{k=1}^{p_e} \lambda_{ek}^{(1)}(c_{ek}.\begin{pmatrix} y \\ n \end{pmatrix} + d_{ek}),$$

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

1. if $\lambda_{e0}^{(1)} > 0$, then $\mathcal{P}_e^{(1)} = \emptyset$;
2. if all $\lambda_{ek}^{(1)} = 0$, then $\mathcal{P}_e^{(1)} = \mathcal{P}_e$.
3. finally, if $\lambda_{e0}^{(1)} = 0$ and $X_e^{(1)} = \{k \mid \lambda_{ek}^{(1)} = 0\}$, then $\mathcal{P}_e^{(1)}$ is a *face* of $\mathcal{P}_e$:

   $$\mathcal{P}_e^{(1)} = \{y \mid c_{ek}.\begin{pmatrix} y \\ n \end{pmatrix} + d_{ek} \geq 0, k \in X_e^{(1)}, c_{ek}.\begin{pmatrix} y \\ n \end{pmatrix} + d_{ek} = 0, k \in [1, p_e] - X_e^{(1)}\},$$

   which may or may not be empty.

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

It is quite obvious that this algorithm is correct and complete. All solutions are 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 is basically an enumeration.

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

Let us go back to program 1, whose DFG is given in Table 1. Each iteration of statement 1 uses a value of `s` which has been produced by the immediately preceding iteration, which is $(1, i, j, k-1)$ if $k \geq 1$ and $(1, i-1, i-1)$ if not.

To construct an affine schedule, let us apply the procedure which has been outlined above. The prototype schedule is:

$$\theta(i,j) = \mu_0 + \mu_1 i + \mu_2(n-i) + \mu_3 j + \mu_4(i-j).$$

Let ϵ_1 (resp. ϵ_2) be a zero-one variable which encodes the fact that edge 1 (resp. 2) is in $U^{(1)}$ or not. Edge 1 is a uniform dependence, which gives simply:

$$\mu_3 - \mu_4 \geq \epsilon_1. \tag{17}$$

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

$$\begin{array}{ll} & \mu_0 + \mu_1 i + \mu_2(n-i) + \mu_3 j + \mu_4(i-j) \\ - & (\mu_0 + \mu_1(i-1) + \mu_2(n-i+1) + \mu_3(i-1)) - \epsilon_2 \\ \equiv & \lambda_0 + \lambda_1(i-1) + \lambda_2(n-i) + \lambda_3 j + \lambda_4(i-j) + \lambda_5(1-j). \end{array}$$

This is equivalent to:

$$\begin{aligned} \mu_1 - \mu_2 + \mu_3 - \epsilon_2 &= \lambda_0 - \lambda_1 + \lambda_5, && (18)\\ \mu_4 - \mu_3 &= \lambda_1 - \lambda_2 + \lambda_4, && (19)\\ \mu_3 - \mu_4 &= \lambda_3 - \lambda_4 - \lambda_5, && (20)\\ 0 &= \lambda_2. && (21) \end{aligned}$$

If $\epsilon_1 = 1$, these conditions are clearly inconsistent. From Equ. (19) and (21) one derives:

$$\mu_4 - \mu_3 = \lambda_1 + \lambda_4 \geq 0,$$

which contradict Equ. (17). We must conclude that program 1 has no affine schedule. This conclusion does not depend on the value of ϵ_2. Hence, the only possibility is setting $\epsilon_1 = 0$. The problem now has, among its solutions:

$$\mu_0 = \mu_2 = \mu_3 = \mu_4 = 0, \mu_1 = 1,$$

i.e., $\theta(i,j)[1] = i$. Now, the only condition for the next component is Equ. (17). We may set $\epsilon_1 = 1$. A possible solution is then $\mu_3 = 1$, $\mu_4 = 0$, or:

$$\theta(i,j)[2] = j.$$

These two results give schedule (2).

4 A greedy algorithm

4.1 Heuristics

The basic objective of the greedy 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 (9). Let $N = \max_S N_{SS}$. Schedule (9) 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 heuristic 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 is greedy in so far as it will try to satisfy as much edges as possible at each stage of the solution, and in that it will never go back on a previous decision. 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 condition (3) is homogeneous and since the latency as given by Equ. (5) 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 suggests the introduction of a set of new variables ϵ_e, $e \in U$ and the resolution of a new problem:

$$\begin{aligned} \sigma \quad = \quad & \max \sum_{e \in U} \epsilon_e, \\ & 0 \leq \epsilon_e \leq 1, \\ & \forall e : y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \geq \epsilon_e. \end{aligned} \tag{22}$$

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

Lemma 5 *The solution of linear program (22) is such that all ϵ_e are either 0 or 1.*

> **Proof** Let $\eta = \min\{\epsilon_e \mid \epsilon_e > 0\}$. Suppose $\eta < 1$. Set $\theta' = \theta/\eta$. Since the delays are linear functions of the schedules, the new delay Δ' satisfies:
>
> $$y \in \mathcal{P}_e \Rightarrow \Delta'_e(y) \geq \epsilon_e/\eta.$$
>
> If $\epsilon_e = 0$, set $\epsilon'_e = 0$. If not, since $\epsilon_e/\eta \geq 1$, set $\epsilon'_e = 1$. The pair θ'_S, ϵ'_e is a solution of linear program (22) and σ has increased, a contradiction. ■

Lemma 6 *The solution of linear program (22) is unique.*

Proof Suppose, a contrario, that we are given two distinct solutions $\theta^{(1)}, \epsilon^{(1)}$ and $\theta^{(2)}, \epsilon^{(2)}$, obviously with the same value σ of the objective function. The pair $\theta' = \frac{\theta^{(1)}+\theta^{(2)}}{2}, \epsilon' = \frac{\epsilon^{(1)}+\epsilon^{(2)}}{2}$ is also a solution of program (22). The ϵ' 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 is the union of the corresponding sets of the initial schedules. If those are distinct, this increases σ, a contradiction. ■

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

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

In case of success, that the resulting multidimensional schedule is a solution of condition (3) 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. We may say that the greedy algorithm is correct but no longer complete. Nevertheless, theorem 2 is still true.

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 [3] 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 statement 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 statements $R \in A$, $S \in B$ such that T_{RS} is true, then all statements of A are before any statement 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 statements of A occurs before all statements of B in the program text is 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 $T_{B_n B_0}$ 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 statements, 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 statements of C.*

> **Proof** Let $R_0, R_1, \ldots, R_n = R_0$ be the statements 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 a statement. Continuing in this way around C, we conclude that there are at least d loops which contain all statements in C.
>
> Suppose that at level $d+1$ these statements 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. ■

Suppose now that we have found k components of a multidimensional schedule, and that the set of unsatisfied edges is $U^{(k)}$. The greedy algorithm will terminate if we can prove that, whatever $U^{(k)}$, the solution of linear program (22) satisfies at least one edge of $U^{(k)}$. It is easy to see that the subscript k only enters the problem through the definition of $U^{(k)}$. To simplify the notation, the next theorem omits k and is stated as if we were computing a one dimensional schedule.

Theorem 8 *Let U be an arbitrary subset of the edges in the DFG of a static control program. Then solving program (22) for U always satisfies at least one edge of U.*

Proof I will prove that linear program (22) always has a feasible solution wich satisfies at least one edge of U. This is true a fortiori for the optimal solution.

Let G be the set of statements 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 is 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 statements 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 seems to 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 ϵ_e, the μ_{Sk} and some of the λ_{ek}. Let u be a vector with ones in positions corresponding to the ϵ_e:

$$\sum_e \epsilon_e = u.\mu.$$

Similarly, let $\phi_S(x)$ be a vector such that the schedule for statement 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 Farkas algorithm, including the constraints $\epsilon_e \leq 1$. The first problem is simply a restatement of program (22):

$$\begin{aligned} \sigma &= \max u.\mu, \\ &\quad \mu \geq 0, \\ &\quad G\mu \geq d. \end{aligned} \tag{23}$$

Each schedule is given by the solution of another problem: extending the notations of Part I, let $\mathcal{G}(\epsilon)$ be the set of affine schedules which satisfies:

$$\forall e : y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \geq \epsilon_e.$$

The best concave schedule is given by:

$$\theta(S,x) = \min_{t \in \mathcal{G}(\epsilon)} t(S,x). \tag{24}$$

which may be translated to:

$$\begin{aligned} \theta(S,x) &= \min \phi_S(x).\mu, \\ &\quad \mu \geq 0, \\ &\quad u.\mu = \sigma, \\ &\quad G\mu \geq d. \end{aligned} \tag{25}$$

Since the solution of linear program (23) is unique, the effect of the second constraint in program (25) is to pin the values of the ϵ_e to their optimum values. Let us set up a third problem, in which m is a new positive parameter:

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

Intuitively, we are interested in what happens when m grows very large. Let Y_1 (resp. Y_2) be the solution of program (23) (resp. program (26)). Each vector is a feasible point for the other problem. Hence:

$$\begin{aligned} u.Y_2 &\leq u.Y_1, & (27)\\ mu.Y_1 - \phi_S(x).Y_1 &\leq mu.Y_2 - \phi_S(x).Y_2. & (28) \end{aligned}$$

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 is possible for a large enough value of m iff $u.Y_2 = u.Y_1 = \sigma$. For a large enough m, solving program (26) will thus be equivalent to solving program (25).

The solution of linear program (26) is found by applying the PIP algorithm to its dual:

$$\begin{aligned} \tau \quad = \quad & \min \nu.d, & (29)\\ & \nu \geq 0,\\ & \nu G \leq mu - \phi_S(x). \end{aligned}$$

The solution for τ is 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 [3], 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 gives the value of σ, and the independent term is $-\theta$. The values of the ϵ_e may be recovered at no extra cost by using *complementary slackness* [5, 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 linear program (26) are

$$\epsilon_e \leq 1.$$

If the corresponding dual solution is nonzero, $\epsilon_e = 1$ 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 considers 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 $\theta(R, x)$ occurs in a constraint for $\theta(S, y)$.

Let the statements to be scheduled be divided in two subsets V_1 and V_2, in such a way that there may be edges of the DFG from elements of V_1 to V_2, but none from V_2 to V_1. Let F_1 be the set of edges whose source and sink belong to V_1 (the *internal edges of* V_1), let F_2 be the internal edges of V_2, and let F_{12} be the edges from V_1 to V_2. We will suppose that all the iteration domains of the program are bounded.

Let us first consider the solution of linear problem (22) for V_1 and F_1, for V_2 and F_2, and for the whole program. Let $\sigma^{(1)}, \theta^{(1)}, \epsilon^{(1)}$, $\sigma^{(2)}, \theta^{(2)}, \epsilon^{(2)}$, and σ, θ, ϵ, be the respective solutions. It is clear that:

$$\forall e \in F_{12} : \epsilon_e = 1.$$

If it where not so, we could build a better schedule:

$$S \in V_1 \Rightarrow \theta'(S, x) = \theta(S, x),$$

$$S \in V_2 \Rightarrow \theta'(S, x) = \theta(S, x) + 1,$$

a contradiction.

The restriction of θ, ϵ to V_1 and F_1, or to V_2 and F_2, is a solution of the associated linear program (22). Hence:

$$\begin{aligned} \sum_{e \in F_1} \epsilon_e &\leq \sigma^{(1)}, \\ \sum_{e \in F_2} \epsilon_e &\leq \sigma^{(2)}. \end{aligned} \tag{30}$$

Conversely, from the solutions in V_1 and V_2, we may build a solution for the whole program:

$$\begin{aligned} S \in V_1 \Rightarrow \theta'(S, x) &= \theta^{(1)}(S, x), \\ S \in V_2 \Rightarrow \theta'(S, x) &= \theta^{(2)}(S, x) + L^{(1)}(n) + 1, \end{aligned}$$

where $L^{(1)}$ is the latency of V_1 as a function of the structure parameters, which exists by lemma I.8. For this solution,

$$\begin{aligned} e \in F_1 \Rightarrow \epsilon'_e &= \epsilon_e^{(1)}, \\ e \in F_2 \Rightarrow \epsilon'_e &= \epsilon_e^{(2)}, \\ e \in F_{12} \Rightarrow \epsilon'_e &= 1. \end{aligned}$$

From this we deduce that:

$$\sigma^{(1)} + \sigma^{(2)} + \text{Card } \mathrm{F}_{12} \leq \sigma,$$

which implies:

$$\sigma^{(1)} + \sigma^{(2)} \leq \sum_{e \in F_1} \epsilon_e + \sum_{e \in F_2} \epsilon_e,$$

and, in conjunction with inequality (30), gives:

$$\begin{aligned} \sum_{e \in F_1} \epsilon_e &= \sigma^{(1)}, \\ \sum_{e \in F_2} \epsilon_e &= \sigma^{(2)}. \end{aligned} \tag{31}$$

As consequence, in the matter of deciding which edges are cut, we may solve the problems in V_1 and V_2 independently.

For the present, the causality condition will be understood as:

$$\forall e : y \in \mathcal{P}_e \Rightarrow \Delta_e(y) \geq \epsilon_e$$

for the ϵ_e which have been computed in the preceding step. θ and $\theta^{(1)}$ will now be the best concave schedule which are found by solving linear program (24) or the equivalent program (25) over the whole program and V_1 respectively.

Let us first consider the case of a statement $S \in V_1$. Since, by an argument similar to the preceding one, any causal affine schedule for the whole program may be restricted to a causal affine schedule for V_1, and a causal affine schedule for V_1 may be extended to a causal affine schedule for the whole program, we deduce that $\theta^{(1)}(S, x) = \theta(S, x)$.

The situation is quite different in V_2. Here, we want to modify the set of schedules over which the minimum is taken to schedules which are causal, affine in V_2, and which verify:

$$y \in \mathcal{P}_e \Rightarrow t(S, y) - \theta^{(1)}(R, h_e(y)) \geq 1 \tag{32}$$

for all edges e from $R \in V_1$ to $S \in V_2$. Let $\theta^{(2)}$ be the solution of this modified problem. Consider a schedule t which is causal and affine over the whole program. Since $\theta^{(1)}$ is less than the restriction of t to V_1, t satisfies inequality (32). Hence:

$$\theta^{(2)}(S, y) \leq \theta(S, y).$$

We cannot, however, prove the reverse inequality. The causal affine schedule t for which the minimum is obtained can be extended to a schedule for the whole program by "gluing" $\theta^{(1)}$ in V_1. However, the result is not affine in general. We conclude that the solution of the modified problem in V_2 is in general better than the solution for the whole program. Equality can be proved in the special case where $\theta^{(1)}$ is affine.

The conclusion is that step by step scheduling can be used since it will give results at least as good as global scheduling. This result holds neither for minimum latency schedules nor for bounded delay schedules. For minimum latency schedules, a counter-exemple is given by Darte et. al. [6, Fig. 7]. For bounded delay schedules, observe that the program in Fig I.2 has no such schedule, while each of its statements has one.

5.1 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 $\theta^{(i)}$ being used if necessary in setting up the constraints for $\theta^{(j)}, j > i$. Since $\theta^{(i)}$ 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 linear program like (29). There is 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 $H_k, k < i$.

As we have seen earlier, external edges are always satisfied. If there are unsatisfied internal edges, one must solve the same problem 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, $\{H_1, \ldots, H_n\}$, ranking them according to the reduced graph of U.
  2. For each $i = 1, \ldots, n$, solve linear program (29).
     (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 [2] is striking. In fact, one may conjecture that **codegen** is a simplified version of **Schedule** in which program (29), 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 do at least as well 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 existing 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 scc decomposition may be activated or inhibited at will. Lastly, when solving the 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 linear program, which is sent to the PIP software. PIP is written in C and has been described elsewhere [7, 8].

6.2 Some examples

Consider again program 2. Table 2 is a list of the statements 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 fails. In fact, one may observe that the basic structure of the back-substitution part is the same as that of program 1. Execution of algorithm **Schedule**, however, is successful. The DFG may be divided into three scc's: $\{3\}$, $\{1,2\}$ and $\{4,5\}$. Statement $\{3\}$ has no predecessor, hence its schedule is zero, which means that it can be executed at the beginning of the program. Statements 1 and 2 have an affine schedule, which is given in table 2. When scheduling statements 4 and 5, one finds that edge 109 cannot be satisfied. Algorithm **Schedule** is called recursively. The unsatisfied edge is a loop from statement 4 to itself. The recursive call builds a second component for the schedule of 4. The result 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

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

Statement	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	i	0	$n - i \geq 0$, $i - 1 \geq 0$
4	i, j	$\begin{pmatrix} 2i + 2n - 4 \\ i + j - 2 - n \end{pmatrix}$	$n - i \geq 0$, $i - 1 \geq 0$, $n - j \geq 0$, $i + j - 2 - n \geq 0$
5	i	$2i + 2n - 3$	$n - i \geq 0$, $i - 1 \geq 0$

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	$\langle 2, n - i, 1 + n - i, j \rangle$	$\langle 4, i, j \rangle$	$4i - 3$	$n - 1 - i \geq 0$
107	$\langle 2, n - i, 1 + n - i, 1 + n \rangle$	$\langle 5, i \rangle$	$4i - 2$	$n - 1 - i \geq 0$
108	$\langle 2, n - i, 1 + n - i, 1 + n - i \rangle$	$\langle 5, i \rangle$	$4i - 2$	$n - 1 - i \geq 0$
109	$\langle 4, i, j - 1 \rangle$	$\langle 4, i, j \rangle$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$i + j - 3 - n \geq 0$
110	$\langle 3, i \rangle$	$\langle 4, i, j \rangle$	$2i + 2n - 4$	$2 + n - i - j \geq 0$
111	$\langle 4, i, n \rangle$	$\langle 5, i \rangle$	1	$i - 2 \geq 0$
112	$\langle 3, i \rangle$	$\langle 5, i \rangle$	$2i + 2n - 3$	$1 - i \geq 0$
113	$\langle 5, 1 + n - j \rangle$	$\langle 4, i, j \rangle$	$2i + 2j - 3 - 2n$	

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

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

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

The schedule of table 2 should be compared to the schedule which is given by theorem 2. The first observation is that while theorem 2 gives many dimensional schedules for statement 1 and 2, the result here is one dimensional. This fact indicates that the elimination phase *has* parallelism. The scheduler notices also that statement 3 is an initialization step which can be executed outright. Both methods are in agreement that statements 4 and 5 need multidimensional schedules, and that they cannot be executed before the last elimination step. In the construction of theorem 2, this is obtained by giving differing constant values to the first component of the schedule. Our scheduler is able to compute the date of the last elimination step: $2n-3$, and to add it as an offset to the schedules for 4 and 5. Notice also that instead of using a third component with constant values 0 and 1, statement 4 and 5 are scheduled respectively at even and odd values of the first component of time. This is a consequence of the tendency of the greedy algorithm to construct schedules with minimal dimension.

Scheduling program 2 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 with the standard operating system tools and are given in seconds. The "setup" column gives the time it takes to construct the linear program (29) (this is the Lisp part of the software), while the "PIP" column gives 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 a large enough number of processors, each operation of the program being executed in unit time. 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:

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

```
      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 i = 1, n
3       s = 0.
        do j = 1, i-1
4         s = s + a(n-i+1, n-j+1)*x(n-j+1)
        end do
5       x(n-i+1) = (a(n-i+1, n+1) - s)/a(n-i+1, n-i+1)
      end do

      end
```

Figure 4: Another version of a Gaussian elimination program

Now, if one forgets about rounding errors, this sum may be computed in any order, for instance:

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

After some rearrangements, one gets the program in Fig. 4, which has a one dimensional schedule: see table 5.

The new schedule has an $O(n)$ latency, which indicates that we have indeed found parallelism in the back substitution phase. Observe that this has been done in two steps. The first one goes from `gosse` to `gosser` by rearranging a

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

Table 5: The schedules for program `gosser`

Statement	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	i	0
4	i, j	$\begin{pmatrix} 1 \\ 2j-1 \end{pmatrix}$
5	i	$\begin{pmatrix} 1 \\ 2ip-2 \end{pmatrix}$

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`

summation. This transformation lies outside the scope of the present method. The second step finds a schedule, which may then be used to build a parallel program. Since the schedule for statement 3 is affine in i, while the one for 4 is affine in j, this results in a kind of loop inversion in an imperfect loop nest. This is beyond Allen and Kennedy's algorithm, and also beyond all algorithms which can be applied only to perfectly nested loops.

The delays of the schedule in table 5 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 the bounded delay condition is not enforced 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.

Obtaining multidimensional schedules does not always indicate that the source program has no parallelism. Consider example 5. Its schedule is given in table 8. Its sequential running time is $O(n^3)$. Statement 1 has a two dimensional schedule, which gives a parallel running time $O(n^2)$, and as a consequence, a

```
        do k = 1,n
          do i = 1,k
            do j = 1,k
1             s(i,j,k) = s(i,j,k-1) +
                  s(i-1,j,k)/x(i-1) + s(i,j-1,k)/x(j-1)
            end do
          end do
2         x(k) = s(k,k,k)
        end do
```

Figure 5: A program with limited parallelism

Statement	Loop counters	Schedule	Domain
1	k, i, j	$\begin{pmatrix} 2k-2 \\ i+j-2 \end{pmatrix}$	$\begin{pmatrix} n-k \\ k-1 \\ k-i \\ i-1 \\ k-j \\ j-1 \end{pmatrix} \geq 0$
2	k	$2k-1$	$\begin{pmatrix} n-k \\ k-1 \end{pmatrix} \geq 0$

Table 8: The schedule of program 5

mean degree of parallelism $O(n)$. This program would have been left invariant by Allen and Kennedy's algorithm.

As a last exemple, consider the programm in Fig. 6 – an extract from the Argonne test suite. The bounded delay schedule for this program is:

$$\begin{aligned} t(1,i) &= 1, \\ t(2,i) &= 0, \end{aligned} \tag{33}$$

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 statements can be executed as separate parallel loops, provided that their order be reversed. The best schedule is slightly more complicated: the dual algorithm has noticed that the flow dependence from `b(i)` in statement 2 to `b(i-1)` in the next iteration of statement 1 does not exists for $i = 2$. Hence, the first iteration of statement 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.

```
      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 6: An exemple from the Argonne test suite

Such schedules always exist and are unique; their only drawback is that they may be 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 tries first for a bounded delay schedule and then, in case of failure, for a best concave schedule. Evaluation of such tactics must wait for 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 linear program (29). 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. The second limitation is in term of an upper bound b on the modulus of the elements of matrix G in program (29). In the worst case, finding the exact solution of this problem 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 induce overflows unless b is 1 or 2 at most. As an irritating consequence, the scheduler may fail for programs with large numerical upper bounds. This happens very seldom. The remedy is to replace these large bounds 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 are programs which surpass the possibilities of our implementation. A possible solution in that case is to implement an

```
do i = n,1,-1
  x(i) = x(2*i) + x(2*i+1)
end do
```

Figure 7: A calculation on a tree

approximate scheduler along the lines of theorem 8. The approximate schedule will satisfy some edges of the DFG. Algorithm **Schedule** may then be applied to a deflated problem.

Another point to note is that the method has a tendency to pack scc's as tightly as the dependences allow. This leads to difficulties in the code generation phase. The resulting code is not suitable to most present day parallel computers, which do not exploit "control parallelism" efficiently. It may be that executing the scc's sequentially is the best solution.

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.

However, there are still many unsolved problems with these scheduling algorithms. The greedy algorithm of section 4 is engineered for the construction of low dimensional schedules; does it achieve the minimum possible dimension, and is low dimension equivalent to low latency? In many cases, one may verify a posteriori that the resulting schedules are in fact free schedules, simply by testing that each operation has at least one unit delay incoming edge. If this criterion is not met, we may be near the free schedule, as is the case of example 6, or far from it, as is the case of example I.6, or of the tree summing example 7 whose best concave schedule is simply:

$$\theta(i) = n - i,$$

while its free schedule is:

$$\tau(i) = \lfloor \log_2 n - \log_2 i \rfloor .$$

While example I.6 can be solved quite easily by dissection of the iteration domain, what are we to do for example 7?

The usual method for constructing a parallel equivalent to a sequential program consists in the application of a list of transformations – loop distribution, loop skewing, loop collapsing, loop interchange – to the original program until the parallel loops are in evidence. Most often, parallelism is obtained only if memory use is also modified by transformations such as array and scalar expansion and privatization. Such an approach suffers from the drawback that the interactions between transformations are very complex, and that there is no clear figure of merit to support a hill climbing paradigm. Hence the attempts to construct integrated frameworks in which the result of applying a given set of transformations is obtained directly. An example of such a framework is Allen and Kennedy's algorithm, which gives directly the result of applying loop distribution and reordering to the original program. Another example is Wolfe and Lam method [9], which, when applied to a perfect loop nest, integrates all transformations expressible as a unimodular change of basis. The present work is another such framework, with the difference that the parallel program is not obtained directly. The schedule may be likened to a blueprint of the parallel program, which can be recovered by polyhedra scanning techniques [10].

The present method integrates all transformations which can be expressed as affine mappings of the iteration space. This set includes all transformations quoted above, the wavefront method, and many new combinations of these. Array expansion transformations are also included provided one starts from a dataflow graph rather than from an ordinary dependence graph. There are many others transformations, like iteration space tiling or communication vectorization, whose primary aim is to handle limited resources, and which are beyond the scope of this paper.

While the algorithm obviates the need for a state space search, its present complexity is not really satisfactory. What is needed here is a combination of problem reduction methods, approximation techniques and schedule combination rules for tackling, at least in an approximate way, life size 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, Hervé Leverge, Alain Darte, and all my friends of the "Techniques de Parallélisation" operation of PRC C^3.

References

[1] Paul Feautrier. Some efficient solutions to the affine scheduling problem, I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348,

October 1992.

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

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

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

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

[6] Alain Darte and Yves Robert. Affine-by-statement scheduling of uniform loop nests over parametric domains. Technical Report 92-16, LIP-IMAG, April 1992.

[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.

[9] M. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452–471, October 1991.

[10] Corinne Ancourt and François Irigoin. Scanning polyhedra with DO loops. In *Proc. third SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 39–50. ACM Press, April 1991.

```
@Article{Feau:92bb,
  author =       "Paul Feautrier",
  title =        "Some Efficient Solutions to the Affine Scheduling
                 Problem, {II}, Multidimensional Time",
  volume =       "21",
  number =       "6",
  journal =      "Int. J. of Parallel Programming",
  month =        Dec,
  pages =        "389--420",
  year =         "1992"
}
```