Scheduling Uniform Loop Nests

Alain Darte et Yves Robert

Laboratoire LIP-IMAG, Ecole Normale Supérieure de Lyon
69364 LYON Cedex 07, France
e-mail: [darte,yrobert]@lip.ens-lyon.fr

September 25, 1996

Abstract

This paper surveys scheduling techniques for uniform loop nests. First we introduce the hyperplane method and related variants. Then we extend it by using a different affine scheduling for each statement within the nest. In both cases we present a new, constructive and efficient method to determine optimal solutions.

1 Introduction

Loop nests lie in the heart of supercompilers-parallelizers for supercomputers. On one hand their importance in terms of applications is evident: in many scientific programs, the time spent in the execution of a couple of loops represents a large fraction of the total execution time, while the potential parallelism of these loops is often very important. On the other hand, the regular and repetitive structure of loop nests greatly facilitates the use of dependence analysis techniques and of scheduling and allocation strategies.

The general problem of finding the optimal scheduling for a task system on a parallel machine is NP-hard (due to the communications), even when assuming an unlimited number of processors. However, in the case of a uniform loop - "uniform" meaning "with a finite number of dependence vectors" until further explained -, it is possible to derive transformations that lead to a parallel scheme of execution while preserving the semantics of the sequential version, and, even more important, to assess the quality of such transformations (i.e., to say how close they are from the optimum).

Consider the following loop nest:

for $i_1 = l_1$ to u_1 do
for $i_2 = l_2$ to u_2 do
$\vdots$
for $i_n = l_n$ to u_n do
    Program P composed of k statements $S_1, \ldots, S_k$

The most widely used technique consists in searching for a timing vector π so as to transform the nest into the following scheme:

for $time = 0$ to $time_{max}$ do
    execute in parallel all $p = (i_1, \ldots, i_n)$ such that $\lfloor {}^t\pi.p \rfloor = time$

This is nothing else than Lamport's hyperplane method [7]. We explain this method and related variants in great details, and we propose an efficient technique for determining an optimal scheduling vector π. From a practical viewpoint, Lamport's method leads to a more efficient implementation than one based upon the free (optimal) schedule. From a theoretical viewpoint, we state the following result that holds in a general framework: in the case of a single statement, the difference between the best linear schedule and the free schedule is bounded by a constant independent of the problem size.

Rather than considering the k statements of program P as a whole block, we can try to schedule them separately. A first technique consists in introducing a different constant for each statement: node p of statement S_i is executed at time $\lfloor {}^t\pi.p + c_i \rfloor$. A second technique consists in using an affine scheduling for each statement: with a different timing vector for each statement, node p of statement S_i is executed at time $\lfloor {}^t\pi_i.p + c_i \rfloor$. In both cases, we propose an efficient method for determining the optimal parameters.

The paper is organized as follows: first we define uniform loop nests (section 2). Section 3 is devoted to the hyperplane method, whose optimality is established in section 4 in the case of a single statement. The two mentioned extensions are dealt with in sections 5 and 6 respectively.

Our last result is a negative one: contrarily to the case of a single statement (or of several statements handled as a single block), the affine-by-statement schedule is not nearly optimal. We derive a simple counterexample where the free schedule outperforms the best affine-by-statement schedule by an order of performance. This negative result should not be over-estimated, since affine-by-statement scheduling is always better than linear scheduling - and the difference is often significant -, while it is not more complex from an implementation viewpoint.

2 Uniform loop nests

A loop nest structure in a sequential program possesses a high degree of implicit parallelism. Therefore it is important for a compiler-parallelizer to be able to take this potential into account. We restrict ourselves to the simplest case, that of uniform loop nests, which is already complex to compile efficiently.

2.1 Perfect loop nest

We choose as a program model the model proposed by Banerjee ([1]), which is the following:

for $i_1 = l_1$ to u_1 do
for $i_2 = l_2(i_1)$ to $u_2(i_1)$ do

for $i_3 = l_3(i_1, i_2)$ to $u_3(i_1, i_2)$ do

$\vdots$

for $i_n = l_n(i_1, i_2, \ldots, i_{n-1})$ to $u_n(i_1, i_2, \ldots, i_n{-}1)$ do

begin

Program P:

{ Statement S_1 }

{ Statement S_2 }

$\vdots$

{ Statement S_k }

end

Here, l_1 et u_1 are constants, while $l_j(i_1, \ldots, i_{j-1})$ (resp $u_j(i_1, \ldots, i_{j-1})$) is the maximum (resp minimum) of a finite number of affine functions of $i_1, \ldots, i_{j-1}$. n is the depth of the nest.

The semantics of this program is given by the sequential execution. This sequential execution induces dependences between iterations. There is a dependence between iteration $S_i(I)$ and iteration $S_j(J)$ if:

- $S_i(I)$ is executed before $S_j(J)$.
- $S_i(I)$ and $S_j(J)$ refer to a memory location M, and at least one of these references is a write.
- The memory location M is not written between iteration I and iteration J.

The dependence vector between iteration $S_i(I)$ and iteration $S_j(J)$ is $d_{(i,I),(j,J)} = J - I$. The loop nest is said to be **uniform** if the dependence vectors $d_{(i,I),(j,J)}$ do not depend neither on I, nor on J, and we denote them simply $d_{i,j}$. We can then represent the loop nest as an oriented graph with k nodes (the statements) linked by edges corresponding to the dependence vectors. The computation domain is represented by the integer points contained within a convex $\{x, Ax \leq b\}$.

Here is a well-known example taken from [13]:

Example 1:

for $i = 0$ to N_1 do

for $j = 0$ to N_2 do

begin

{ Statement S_1 } $a(i, j) = b(i - 3, j - 5)$

{ Statement S_2 } $b(i, j) = a(i - 2, j - 4)$

end

In this example, the computation domain is a rectangle: $Dom = \{(i, j) \in Z^2, 0 \leq i \leq N_1, 0 \leq j \leq N_2\}$. The variable $a(i, j)$ is produced by statement $S_1(i, j)$ and consumed by statement $S_2(i + 2, j + 4)$, hence a dependence from S_1 to S_2 of vector

$(2,4)$. Similarly, there is a dependence from S_2 to S_1 with vector $(3,5)$. Both dependences are uniform, and we define the matrix

$$D = \begin{pmatrix} d_1 & d_2 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

to represent them.

We point out that all dependence vectors of a loop nest are lexicographically positive (their first nonzero component is greater than 0), due to the semantics of the sequential execution.

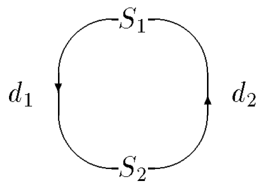

Figure 1: Uniform dependence graph of Example 1

2.2 Free scheduling versus other choices?

If we aim at minimizing the execution time (without taking inter-processor communications into account), the best parallel execution of the loop nest is based upon the free schedule: each instance of each statement is executed as soon as possible. The basic principle is quite simple: we execute at time 0 all nodes in the computation domain without any predecessor. Then at each time-step we execute all nodes whose predecessors have all been already executed [1].

However, such an implementation would be totally inefficient as soon as communication time is not negligible: we lose time when transmitting data single item per single item, we also lose time when scrolling to check whether each consumed data is ready. And how can we bound the number of required processors ? We might choose to have one processor per computation point so as to manage to keep an acceptable execution time.

Another approach has emerged from the work done on compiling FORTRAN loops. The idea is to keep the structural regularity of the original loops and to apply regular transformations to extract some parallelism. These transformations are performed at compile-time and they should render the implicit parallelism hidden in the loop nest fully explicit. This is the key to implementing some optimizations before execution: these optimizations [5] aim at reducing the communication time (e.g. through message vectorization), or at decreasing the number of synchronizations (e.g. using structural information to eliminate some tests), or at diminishing memory access costs (through cache re-use or tuned allocation strategies). The loop nest structure is the simplest structure for applying such transformations. It is therefore of paramount importance to fully understand before tackling the task of designing a FORTRAN compiler-parallelizer.

[1] It is a list scheduling technique "at earliest time" with an unlimited number of processors.

3 The hyperplane method

Points of the computation domain non linked by dependence vectors can be executed simultaneously. The most natural idea is to determine a scheduling vector π and a family of affine hyperplanes $H(t)$ such that the set of points executed at a time t be equal to the intersection of the domain with the hyperplane $H(t)$. The flow of computations goes from $H(t)$ to $H(t+1)$ by a translation of vector π; hence the name of the method that we describe in this section: the computations progress like a wavefront parallel to the family of hyperplanes $H(t)$.

3.1 Lamport's method

One of the possible optimizations on a loop nest is to try to transform it into an equivalent nest for which a number of internal loops can be executed in parallel. When all the internal loops except the first one are parallel, the nest reduces to an expression of the form:

```
for time = time_0 to time_N do
    for p ∈ E(time) do in parallel
        P(p)
    enddo
enddo
```

The external loop corresponds to an iteration time and $E(time)$ is the set of all the points computed at the step $time$. The search for a scheduling vector allowing to obtain the form above is complex. In order to be sure that the new loop nest computes the same thing as the original nest, the direction of the dependences has to be preserved. Lamport [7] proposes to simplify the problem and suggests to restrict the search to linear scheduling functions. We will see that the regularity of the domain and of the dependence vectors can justify such an intuitive choice.

In the case of a uniform loop nest, the search for an admissible linear function can be done by looking for a vector π with rational components such that for each dependence vector d, we have ${}^t\pi.d \geq 1$. The set $E(time)$ of the points computed at the step $time$ is defined by

$$E(time) = \{p; Ap \leq b; \lfloor {}^t\pi.p \rfloor = time\}$$

Actually, $E(time)$ is not always an hyperplane: it can be constituted by many successive parallel fronts.

Since all dependence vectors are lexicographically positive, it is easy to build an admissible scheduling vector ([7, p.88] or [3, p.165]): let $D = (d_1, \ldots, d_m)$ be the matrix of dependence vectors, of size $n \times m$, where n is the depth of the nest and m is the number of dependence vectors. We assume that the columns of D have been sorted according to the lexicographic order. Let k_1 be the index of the first nonzero component of d_1, the "smallest" vector of D. Since d_1 is positive, $d_{1,k_1} \geq 0$. Take $\pi_{k_1} = 1$ and $\pi_k = 0$ for $k_1 < k \leq n$. Before defining the first components of π, remark that whatever their values can be, ${}^t\pi.d_1 > 0$. Now, let k_2 be the index of the first nonzero component of d_2. Because d_1 is lexicographically smaller than d_2, $k_2 \leq k_1$. Let $\pi_k = 0$ for $k_2 < k < k_1$ and take for π_{k_2} the smallest positive integer such that ${}^t\pi.d_2 > 0$, modifying maybe π_{k_1} if $k_2 = k_1$. Continuing the process, we obtain an admissible vector.

Many loop transformation methods have been proposed, and almost all of them amount to looking at a particular class of scheduling vectors. Among these, the synthesis method of systolic arrays has permitted to bring to the fore the importance of linear scheduling functions.

3.2 The synthesis method of systolic arrays

Karp, Miller et Winograd's works [6] has permitted to better understand the structure of the parallelism contained in uniform recurrence equations. Their aim was to study the link between the different important parameters that are: execution time, potential parallelism, number of processors needed. One of their results, which contributed to the birth of the synthesis method called projection method, is the following: consider a uniform recurrence equation on the domain

$$F_n = \{j \in Z^n, j_i \geq 1,\ i = 1, \ldots, n\}$$

with dependence vectors $d_i, i = 1, \ldots, m$. Karp, Miller and Winograd have shown how to link the existence of a schedule for all each points in F_n with the solution of the two dual problems:

$$I \left\{ \begin{array}{l} j - \sum_{i=1}^{m} u_i d_i \geq 0 \\ u_i \geq 0, i = 1, 2, \cdots, m \\ \max \sum_{i=1}^{m} u_i \end{array} \right. \qquad II \left\{ \begin{array}{l} d_i.x \geq 1, i = 1, 2, \cdots, m \\ x_i \geq 0, i = 1, 2, \cdots, n \\ \min j.x \end{array} \right.$$

A schedule exists if and only if for all $j \in F_n$, the two dual problems have a common solution (denoted by $m(j)$). All dependence paths that remain in F_n and terminate in j lead to a feasible solution to problem I, hence the time of the free schedule (denoted by $\sigma_{free}(j)$) is larger than $m(j)$. Furthermore, there exists a constant K such that $m(j) - \sigma_{free}(j) \leq K$ for all those points $j \in F_n$ that are not "too close" to the domain boundary. Problem II can be interpreted in terms of finding for each point $j \in F_n$ the optimal linear schedule vector $x(j)$ that will lead to execution of point j at time $m(j)$. Such a vector $x(j)$ is an extremal point of the domain

$$\{x \in Q^n,\ d_i.x \geq 1 \,\mathrm{for}\, 1 \leq i \leq m,\ x_i \geq 0 \,\mathrm{for}\, 1 \leq i \leq n\}$$

Hence there is a finite number of such vectors and we see that this strategy leads to piecewise linear schedules, the same vector x being the optimal linear vector over a whole subregion of F_n. Comparing the two dual problems has permitted to show the nearly-optimality of piecewise linear schedules on the domain F_n.

Justified by this result, the projection method proposed by Quinton [14] consists in representing the systolic algorithm as a uniform computation domain, in scheduling it with a linear function and in projecting it on what will be the systolic array. The system of recurrence equations is considered as one multi-dimensional equation applied to a domain $\{x, Ax \leq b\}$, and the hyperplane method is used on this domain. A natural question arises: the result of Karp, Miller et Winograd dealed with piecewise linear functions and not linear functions. What can be said about the optimality of linear functions? Furthermore, how can we find the best vector π?

The answer to these questions is important because the regularity coming from the use of linear scheduling functions seems, in view of the systolic results, to be very interesting for the rewriting of loop nests. Despite its simplicity, the systolic model has at least permitted, as a computation model, to better understand the case of simple loops, a necessary step before tackling the compilation of more complicated loops.

3.3 Related methods

For uniform loop nests, many particular cases of the hyperplane method have been proposed, in particular when there are cycles in the reduced graph of dependence (as for example 1).

Selective shrinking This technique (see [13]) consists in looking at the dependence matrix $D = (d_1, \ldots, d_m)$ row by row. As soon as a row i is found whose m components $d_{i,j}$ are all nonzero, the vector $\pi = \frac{1}{\min(d_{i,j}, 1 \leq j \leq m)} e_i$ is selected, where e_i is the i-th basis vector.

True dependence shrinking This technique (see [13, 12]) consists in computing for each dependence vector its "true distance", defined as the number of steps (in a sequential mode) between the evaluation of two dependent statements. For a loop nest whose domain is a parallelipiped

$$\{0 \leq i_1 \leq N_1, 0 \leq i_2 \leq N_2, \ldots, 0 \leq i_n \leq N_n\}$$

we obtain

$$\text{true\_distance}(d_i) = d_{n,i} + d_{n-1,i} * (N_n + 1) + \cdots + d_{1,i} \prod_{j=2}^{n} (N_j + 1)$$

where $d_i = ^t (d_{1i}, d_{2i}, \ldots, d_{ni})$. Then, one selects

$$\pi = \frac{(\prod_{j=2}^{n}(N_j+1), \prod_{j=3}^{n}(N_j+1), \ldots, 1)}{\min(\text{true\_distance}(d_i), 1 \leq i \leq m)}$$

as scheduling vector.

Unimodular transformations Techniques summarized in [22] aim at exchanging two loops ("loop interchange") or at applying a skewing of a loop by another ("loop skewing"). These transformations are unimodular: the corresponding matrix is $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ or $\begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix}$. The success of such transformations is equivalent to the possibility to choose a basis vector e_i or a vector $e_i + \lambda e_j$ as scheduling vector.

4 Optimality for the case of a single statement

The theory of linear programming which was used by Karp, Miller and Winograd, gives an insight to the link between the execution time given by an optimal linear schedule and the length of the longest path of dependences, i.e. the time of the free schedule. Furthermore, it permits to show that, for a single statement and a convex computation domain, linear scheduling is nearly optimal. Lastly, it gives a constructive way of finding the optimal scheduling vector π.

4.1 Nested loops and uniform recurrence equations

As said above, we can consider a uniform loop nest as a set of vertices (the statements) linked by dependence constraints (given by the dependence vectors), where each vertex represents the whole domain $\{x, Ax \leq b\}$ because of the uniformity of the dependences. This structure is similar to the structure of reduced graphs for uniform recurrence equations (see [6] or [17]) except that, in a reduced graph extracted from a loop nest, the dependence vectors are lexicographically positive. The semantics of a loop nest is given by the sequential execution whereas the semantics of uniform recurrence equations is implicit, the right-hand side being computed before the left-hand side.

In the following, we express uniform loop nests in terms of uniform recurrence equations, and we study the simplest case of a single equation with unit time of computation (called "uniform dependence algorithm" (UDA) by Fortes ([21])). Only the structural information of the UDA is important, i.e:

- The computation domain represented linearly by the set of points which satisfy $Ax \leq b$.
- The dependence vectors represented by the matrix D.

Denote by T_f the time of the free schedule (the length of the longest path of dependences between points of the domain) and by T_l the execution time given by the best linear schedule. We summarize here the results shown in [2].

4.2 Optimality of linear scheduling

To simplify the notations we write the scheduling vectors as row vectors. A point p in the domain will be executed at time $T(p) = \lfloor \pi.p \rfloor$. The search for the best scheduling vector can be done by minimizing the execution time, i.e. $\max_{(Ap \leq b, Aq \leq b)}(\lfloor \pi.p \rfloor - \lfloor \pi.q \rfloor)$ for integer points p and q. One can approximate the solution by considering the optimal solution π of the associated continuous problem:

$$\min_{(XD \geq 1)} \max_{(Ap \leq b, Aq \leq b)} X(p - q)$$

The optimal value of this problem is equal to the maximum value of

$$\begin{cases} y \geq 0 \\ Ap \leq b \\ Aq \leq b \\ Dy = p - q \\ \max \sum_{i=1}^{m} y_i \end{cases}$$

which can be interpreted as the length of the longest path given by a linear combination (with rational and positive components) of dependence vectors, from a point q to a point p, both belonging to the domain of computation.

The existence of very thin domains for which the difference between the time of the free schedule and the time of linear scheduling is not bounded (see [6]) leads us to make a realistic hypothesis on the shape of the domain.

Definition 1 *A domain P is* ***fat*** *if*

(1) $0 \in P$

(2) $0 + d_i \in P$, $\forall i, 1 \leq i \leq m$

(3) $0 + e_i \in P$, $\forall i, 1 \leq i \leq n$, *where the e_i's are the canonical basis vectors.*

Now, considering a fat domain $\{Ax \leq b\}$ and the UDA over the family of domains $\{Ax \leq Nb\}$ enables us to parametrize the domain size as a function of N. N is the parameter size.

Definition 2 *The* ***fatness*** *of P is $t = \max\{\tau > 0; \frac{1}{\tau}P$ is fat$\}$ if it exists.*

With these two definitions, we can state the two following theorems whose demonstration can be found in [2]:

Theorem 1 *For a UDA algorithm on a domain J of dimension n and with fatness t greater than $3n+1$, $1 \geq T_f/T_l^* \geq 1 - O(n/t)$ where $T_l^* \geq T_l - 1$. The relative error between the optimal linear schedule and the free schedule is bounded by $O(n/t)$.*

Now, consider the UDA over the family of domains $\{Ax \leq Nb\}$ where $\mathcal{D} = \{Ax \leq b\}$ has fatness 1.

Theorem 2 *Let n be the problem dimension. Then*

$$T_l - T_f \leq 1 + (3n+1)t_0$$

for $N \geq 3n+1$, where t_0 is the time of the optimal linear schedule for the unit domain $J \equiv (A, b)$. The optimal linear schedule and the free schedule differs by a constant independent of the size of the domain. The constant that we give only depends upon the shape of the domain (the constraint matrix A and the vector b) and upon the dependence vectors (the dependence matrix D).

These results give a justification of the hyperplane method for the case of a single equation, and of the projection method for the synthesis of systolic arrays. Indeed, in this second method, the dependence graph is first uniformized, then the whole system of equations is considered as a single multi-dimensional equation. Thus, for such a model, the use of linear scheduling is justified.

4.3 The search for the best scheduling vector

Fortes et Shang ([21]) propose a complex method to obtain the optimal scheduling vector of the continuous problem, based upon a partition of the solutions space into subcones on which they solve a linear fractional problem. Lisper ([9]) proposes an approach in which all couples of extremal points of the domain are enumerated before solving a linear program for each of them. Here, we propose a conceptually simpler method which consists in solving only one single linear program.

Finding the optimal scheduling vector is solving the problem:

$$\min_{(XD\geq 1)} \max_{(Ap\leq b, Aq\leq b)} X(p-q)$$

But, by duality,

$$\begin{cases} Ap \leq b \\ Aq \leq b \\ \max X(p-q) \end{cases}$$

is equal to:

$$\begin{cases} X_1 A = X \\ X_2 A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ \min\ (X_1 + X_2) b \end{cases}$$

the existence of the maximum implying the existence of the minimum. Thus, the search for the optimal scheduling vector can be done by solving the following linear problem:

$$\begin{cases} XD \geq 1 \\ X_1 A = X \\ X_2 A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ \min\ (X_1 + X_2) b \end{cases}$$

Remark that this problem is linear in b. The search for the best scheduling vector over the family of domains $\{Ax \leq Nb\}$ is reduced to the search on the domain $\{Ax \leq b\}$ which can be done without knowing the parameter N, thus at compile-time. For domains with many parameters, one can use a parametrized simplex algorithm like PIP ([4]).

Example 2 [21, p.729]: In this example, the computation domain is a quadrilateral given by:

$$A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ -1 & -1 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ 2N \\ -N \end{pmatrix}$$

and the dependence vectors are:

$$D = \begin{pmatrix} d_1 & d_2 & d_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 2 \\ -3 & 4 & 0 \end{pmatrix}$$

Solving the associated linear problem leads to $\pi = (\frac{7}{10}, -\frac{1}{10})$ which is, as expected, the vector found by Shang and Fortes' method.

5 Adding a scheduling constant to each statement

We have seen that the best linear schedule is nearly optimal for uniform loop nests whose program P is composed of a single statement, and whose computation domain is regular enough. A natural question arises: does this result generalize to the case of several statements ? The answer is negative, and we extend the linear scheduling technique through several refinements. In this section, we study the impact of introducing a different constant for each statement. We show that this simple modification permits to retrieve and to improve the results of the Index Shift Method [10, 19] for cycle shrinking.

5.1 The Index Shift Method

The Index Shift Method (ISM) [10]) is best explained with the following example, taken from [12]:

Example 3:

for $i = 0$ to N do
for $j = 0$ to N do
begin
{ Statement S_1 } $a(i,j) = b(i,j-6) + d(i-1,j+3)$
{ Statement S_2 } $b(i+1,j-1) = c(i+2,j+5)$
{ Statement S_3 } $c(i+3,j-1) = a(i,j-2)$
{ Statement S_4 } $d(i,j-1) = a(i,j-1)$
end

The dependences are the following:

$$\begin{aligned}
S_1 \longrightarrow S_3 : d_1 &= \begin{pmatrix} 0 \\ 2 \end{pmatrix} \\
S_3 \longrightarrow S_2 : d_2 &= \begin{pmatrix} 1 \\ -6 \end{pmatrix} \\
S_2 \longrightarrow S_1 : d_3 &= \begin{pmatrix} 1 \\ 5 \end{pmatrix} \\
S_1 \longrightarrow S_4 : d_4 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\
S_4 \longrightarrow S_1 : d_5 &= \begin{pmatrix} 1 \\ -4 \end{pmatrix}
\end{aligned}$$

We obtain the dependence matrix:

$$D = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & -6 & 5 & 1 & -4 \end{pmatrix}.$$

The computation domain is a square: $Dom = \{(i,j) \in Z^2, 0 \le i,j \le N\}$. There are two dependence cycles, namely $C_1 = (S_1, S_3, S_2)$ and $C_2 = (S_1, S_4)$.

Let us first compute the best scheduling vector using the previous method: we find $\pi = (7,1)$ and an execution time equal to $8N$. Next, draw a directed graph (Figure 2) whose nodes are the statements S_1 to S_4 and whose edges are weighted by the values $e_i = \pi d_i$.

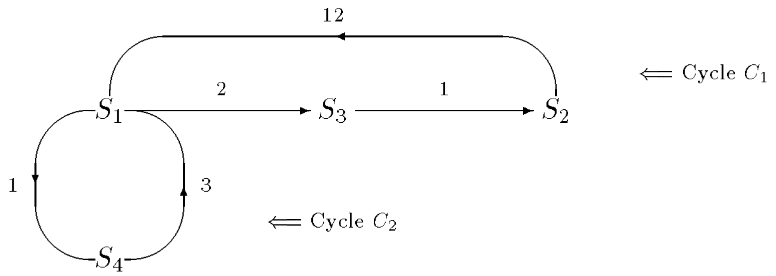

Figure 2: Dependence cycles of Example 3

The idea of the Index Shift Method (ISM) is to apply some retiming to this graph. Indeed, consider cycle C_2 with the edge $e_{14} = (S_1, S_4)$ weighted by $w_{14} = 1$ and the reverse edge $e_{41} = (S_4, S_1)$ weighted by $w_{41} = 3$. Equalizing both weights to the value 2 would permit to choose $\frac{1}{2}\pi$ as a new scheduling vector, since the condition $\frac{1}{2}\pi.d \ge 1$ would be preserved for the dependence vectors of cycle C_2. Similarly, $w_{21} = 12$ and we can shift part of this weight to the other edges of cycle C_1 so that the smallest weight of all edges becomes equal to 2, thereby enabling the choice of $\frac{1}{2}\pi$ as a scheduling vector for the nest.

Systolicians will have recognized the retiming procedure of Leiserson and Saxe [8]. Generally speaking, given a scheduling vector π such that $\pi.D \ge 1$, let $Cycl$ denote the set of all cycles in the dependence graph whose nodes are the statements $S_1 \ldots S_k$ and where each edge (S_i, S_j) corresponding to the dependence vector d_{ij} is weighted by πd_{ij}. For a cycle C in $Cycl$, let $T(C)$ be its total weight and $k(C)$ its length. The ISM method aims at replacing π by the new vector

$$\frac{1}{\min(\frac{T(C)}{k(C)}; C \in Cycl)}\pi$$

Let S_i^k be the transformation that consists of adding k to the weights of all incoming edges to node S_i, and subtracting k from the weights of all outgoing edges from S_i. In our example, the transformation $S_2^1 S_4^1$ leads to the graph of figure 3 and does permit to choose $\frac{1}{2}\pi$.

What does it mean to apply a transformation S_i^k to the graph for the loop nest, and how the range of the loop indices is affected? First of all, applying the transformation S_i^k amounts to replacing the execution of a given instance $S_i(p)$ of statement S_i by the execution of another instance $S_i(p')$ such that the scheduling difference between the two instances is equal to $k = \pi(p' - p)$. To determine p', we only need to know a vector u such that $\pi.u = 1$ and we let $p' = p - ku$. Moreover, the vector u enables us to compute the new bounds for the loop indices. In the example the transformation S_2^1 amounts to shifting index j by one unit in statement

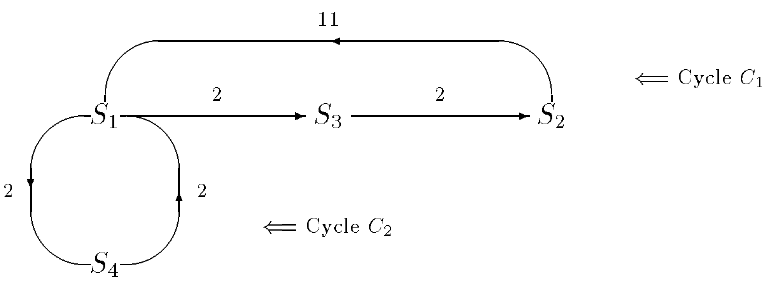

Figure 3: New weights obtained by $S_2^1 S_4^1$

S_2. But as j moves from 0 to N in statement S_1, the total range is extended from 0 to $N+1$. Some iteration instances will be meaningless for some statements: in our example this is the case when $j = N+1$ (statements S_1 and S_3) or when $j = 0$ (statements S_2 and S_4). Applying this process, we derive the new loop nest:

for $i = 0$ to N do
for $j = 0$ to $N+1$ do
begin
$\{S_1\}: \quad a(i,j) = b(i, j-6) + d(i-1, j+3)$
$\{S_2\}: \quad b(i+1, j-2) = c(i+2, j+4)$
$\{S_3\}: \quad c(i+3, j-1) = a(i, j-2)$
$\{S_4\}: \quad d(i, j-2) = a(i, j-2)$
end

Should we search for a scheduling vector for this new nest from scratch, we would find $\pi = (\frac{7}{2}, \frac{1}{2})$ and an execution time approximately equal to $4N$, nearly halving the execution time derived with the best linear schedule.

5.2 Scheduling with parallel hyperplanes

Of course there is no reason *a priori* for the application of the ISM method to the best linear scheduling vector to give the best execution time. May be we can achieve a smaller execution time by applying the ISM method to an arbitrary scheduling vector ? To see this, we return to Example 3: let $\pi = (a, b)$ be an arbitrary scheduling vector. Necessarily, $b \geq 1$ and $a \geq 6b+1$, to fulfill the constraint $\pi.D \geq 1$. We get $\min(\frac{T(C)}{k(C)}; C \in Cycl) = \lfloor \frac{a-3b}{2} \rfloor$. Letting $\pi = (2c+1, 1)$ $(c \geq 3)$ and applying the transformation S_1^{2-c}, we obtain an execution time equal to $2\frac{c+1}{c-1}N$, which again is half of the best previous result ! In fact there are examples [19] showing that it is possible to gain an arbitrary factor over the optimal linear schedule.

Reasoning in terms of scheduling techniques, we can view the ISM method as a natural generalization of the hyperplane method which consists in introducing a constant for each statement: the instance p of statement S_i is executed at time $\pi p + c_i$; we keep a unique scheduling vector but we align statements the ones with the others (see [15]).

Let us state this formally: let $S_1, \ldots, S_k$ be the k statements of a uniform loop nest, and let

- D_i be the matrix representing the self-dependences for the statement S_i.
- $d_{i,j}$ be the dependence vector linking the statement S_i to the statement S_j: the instance p of S_j depends on the instance $p - d_{i,j}$ of S_i. For the sake of simplification, we assume that $d_{i,j}$ is unique whenever it exists.
- $\lfloor \pi p + c_i \rfloor$ be the execution time of $S_i(p)$.

The scheduling vector π must satisfy $\pi.D_i \geq 1$ for all i. The dependence condition linking statement S_i to statement S_j translates into: for each index p of the computation domain $\{x; Ax \leq b\}$, $\lfloor \pi p + c_j \rfloor - \lfloor \pi(p - d_{i,j}) + c_i \rfloor \geq 1$, that is $\pi.d_{i,j} + c_j - c_i \geq 1$.

Moreover, since all the statements are scheduled by the same vector, they all have the same delay $\mathrm{delay}(\pi) = \max_{(Ap \leq b, Aq \leq b)} \pi(p - q)$, and the total execution time is given by $\mathrm{delay}(\pi) + \max_{i,j}(c_i - c_j)$. We neglect $\max_{i,j}(c_i - c_j)$ (which does not depend on the domain size) in front of the delay, which is a linear function of N over the domain $\{x, Ax \leq Nb\}$. Using the method of section 4.3, we have to solve the following linear problem:

$$\begin{cases} \forall i, XD_i \geq 1 \\ X_1 A = X \\ X_2 A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ \\ \forall (i,j) \text{ such that } d_{i,j} \text{ exists} \\ X d_{i,j} + c_j - c_i \geq 1 \\ \min\, (X_1 + X_2) b \end{cases}$$

Going back to Example 3, we obtain $\pi = (\frac{11}{7}, -\frac{1}{7})$, $c_1 = -\frac{9}{7}$, $c_2 = -\frac{10}{7}$, $c_3 = 0$ and $c_4 = -\frac{1}{7}$, leading to a total execution time of the order of $\frac{12}{7}N$. We point out that the improvement over the ISM method is due to the larger search space that we leave for finding a scheduling vector π when determining it simultaneously with the constants, instead of *a priori*: indeed the constraints $\pi.d_{i,j} \geq 1$ no longer need to be satisfied.

6 Affine scheduling functions for each statement

Consider the very simple example of the uniformized matrix-matrix product whose reduced graph is shown figure 4. The computation domain is a cube of size n.

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            S1 : a(i, j, k) = a(i, j - 1, k)
            S2 : b(i, j, k) = b(i - 1, j, k)
            S3 : c(i, j, k) = c(i, j, k - 1) + a(i, j - 1, k) * b(i - 1, j, k)
        enddo
    enddo
enddo
```

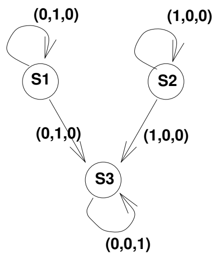

Figure 4: The reduced graph of the matrix-matrix product

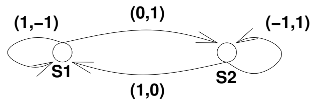

Figure 5: The reduced graph of a fully sequential example

The optimal time given by a linear function is $3n$ because of the constraint $\pi D \geq 1$ whereas the length of the longest path is $2n$. A solution is to schedule the equations as follows:

- The computation of S_1 at point p is carried out at time $(0, 1, 0).p$.
- The computation of S_2 at point p is carried out at time $(1, 0, 0).p$
- The computation of S_3 at point p is carried out at time $(0, 0, 1).p + n$

We check that such a schedule satisfies the dependence constraints and gives an execution time of $2n$. Actually, we have looked for a solution among the *affine-by-statement* schedules. These schedules offer the same advantages as linear schedules. Their implementation is quite similar, one has just to consider the computation domain as k (number of statements) identical computation domains synchronized by the scheduling functions.

Consider the more complicated fully sequential example (not coming from a nested loop however) inspired by Karp, Miller et Winograd's example ([6, p.577]) and whose reduced graph is given by figure 5. The longest path of dependences (on a square of size n) is of length n^2 (the surface of the square). The following schedule:

- The computation of S_1 at point p is carried out at time $(n + 1, n).p$.
- The computation of S_2 at point p is carried out at time $(n, n + 1).p$.

gives an execution time of $n^2 + o(n^2)$ and thus is nearly optimal. One can think that a similar theorem as the theorem of optimality for one statement could be stated for a system of equations and affine-by-statement scheduling. Unfortunately, this is not true, as shown by the counterexample that we give at the end of this section. However, to be able to find automatically the best affine-by-statement schedule is interesting because it often gives quite good schedules which are very difficult to find by hand.

6.1 Expression of the dependence constraints

As for the linear schedules, we deal with the continuous problem. With each equation i is associated an affine scheduling function $X_i.p + c_i$ (see [11]. To simplify, we will assume that statement i and statement j are linked by only one dependence vector denoted by $d_{i,j}$ and that each statement has self-dependences represented by the matrix D_i. Then, the dependence constraints can be written as follows:

$$\left\{\begin{array}{l}\text{Self-dependences:}\\ \forall i, 1 \le i \le k, X_i D_i \ge 1\\ \\ \text{Dependences to } i \text{ from each of its predecessors } j:\\ \forall p, Ap \le b\\ \forall i, 1 \le i \le k\\ \forall j \in Pred(i)\\ (X_i - X_j)p + X_j d_{j,i} + c_i - c_j \ge 1\end{array}\right.$$

Looking for the best affine-by-statement schedule is equivalent to solving the following problem:

$$\left\{\begin{array}{l}\forall i, 1 \le i \le k, X_i D_i \ge 1\\ \\ \forall p, Ap \le b\\ \forall i, 1 \le i \le k\\ \forall j \in Pred(i)\\ (X_i - X_j)p + X_j d_{j,i} + c_i - c_j \ge 1\\ \\ \min \max_{(Ap \le b, Aq \le b)} \max_{(i \in [1\ldots k], j \in [1 \ldots k])} (X_i p + c_i - X_j q - c_j)\end{array}\right.$$

We are going to simplify this problem. The following lemma let us better understand why the constraints that we are going to add are not restrictive but redundant.

Lemma 1 *Let X be any scheduling vector for which the execution time on the domain $\{x, Ax \le b\}$ is bounded. Then, there exists a vector $X' \ge 0$ such that $X = X'A$.*

Proof : If there is no such vector X', then by Farkar's lemma ([20, p.89]), there exists a vector y such that $Ay \ge 0$ and $Xy < 0$. Let p_0 be a point of the domain (assumed non-empty). Consider the points $p_\lambda = p_0 - \lambda y$ for $\lambda \ge 0$. They all belong to the domain (the domain has a ray in the direction y). Then, the execution time goes to ∞ when λ goes to ∞. □

This shows that we restrict the search to scheduling vectors satisfying:

$$\begin{cases} \forall i, 1 \leq i \leq k \\ X_i = X_i^{'} A \\ X_i = -X_i^{''} A \\ X_i^{'} \geq 0 \\ X_i^{''} \geq 0 \\ \\ \forall i, 1 \leq i \leq k \\ \forall j \in Pred(i) \\ X_j - X_i = X_{j,i} A \\ X_{j,i} \geq 0 \end{cases}$$

furthermore:

$$\begin{cases} \forall p, Ap \leq b \\ \forall i, 1 \leq i \leq k \\ \forall j \in Pred(i) \\ (X_i - X_j)p + X_j d_{j,i} + c_i - c_j \geq 1 \end{cases}$$

can be rewritten:

$$\begin{cases} \forall i, 1 \leq i \leq k \\ \forall j \in Pred(i) \\ -(\max_{(Ap \leq b)}(-X_i + X_j)p) + X_j d_{j,i} + c_i - c_j \geq 1 \end{cases}$$

then

$$\begin{cases} \forall i, 1 \leq i \leq k \\ \forall j \in Pred(i) \\ -X_{j,i} b + X_j d_{j,i} + c_i - c_j \geq 1 \end{cases}$$

because

$$(\max_{(Ap \leq b)} (-X_i + X_j)p) = (\min_{(X_{j,i} A = X_j - X_i, X_{j,i} \geq 0)} X_{j,i} b)$$

Furthermore

$$\max_{(Ap \leq b, Aq \leq b)} \max_{(i \in [1 \dots k], j \in [1 \dots k])} (X_i p + c_i - X_j q - c_j)$$

can be successively rewritten

$$\max_{(i \in [1 \dots k], j \in [1 \dots k])} \max_{(Ap \leq b, Aq \leq b)} (X_i p + c_i - X_j q - c_j)$$

$$\max_{(i \in [1 \dots k], j \in [1 \dots k])} (c_i - c_j) + \max_{(i \in [1 \dots k])} \max_{(Ap \leq b)} (X_i p) + \max_{(j \in [1 \dots k])} \max_{(Aq \leq b)} (-X_j q)$$

$$\max_{(i \in [1 \dots k], j \in [1 \dots k])} (c_i - c_j) + \max_{(i \in [1 \dots k])} \min_{(X_i' A = X_i, X_i' \geq 0)} (X_i' b) + \max_{(j \in [1 \dots k])} \min_{(X_j'' A = -X_j, X_j'' \geq 0)} (X_j'' b)$$

$$\max_{(i \in [1 \dots k], j \in [1 \dots k])} (c_i - c_j) + \min_{(X_i' A = X_i, X_i' \geq 0)} \max_{(i \in [1 \dots k])} (X_i' b) + \min_{(X_j'' A = -X_j, X_j'' \geq 0)} \max_{(j \in [1 \dots k])} (X_j'' b)$$

because these constraints are independent, then

$$\min_{(X_i' A = X_i, X_i' \geq 0)} \max_{(i \in [1 \dots k])} (X_i' b + c_i) + \min_{(X_j'' A = -X_j, X_j'' \geq 0)} \max_{(j \in [1 \dots k])} (X_j'' b - c_j)$$

This leads to the unique linear problem:

$$\left\{\begin{array}{l}
\text{Constraints for each statement:}\\
\forall i, 1 \le i \le k\\
X_i = X_i' A\\
X_i = -X_i'' A\\
X_i' \ge 0\\
X_i'' \ge 0\\
X_i D_i \ge 1\\
\\
\text{Constraints between statements:}\\
\forall i, 1 \le i \le k\\
\forall j \in Pred(i)\\
X_j - X_i = X_{j,i} A\\
X_{j,i} \ge 0\\
-X_{j,i} b + X_j d_{j,i} + c_i - c_j \ge 1\\
\\
\text{Constraints coming from the objective function:}\\
\forall i, 1 \le i \le k\\
\gamma_1 - X_i' b - c_i \ge 0\\
\gamma_2 - X_i'' b + c_j \ge 0\\
\\
\min(\gamma_1 + \gamma_2)
\end{array}\right.$$

This linear problem permits to find the best affine-by-statement schedule. For example, for the matrix-matrix product on a n-by-n square, it gives the solution:

$$\begin{array}{lll}
X_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} & X_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} & X_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
c_1 = -n & c_2 = -n & c_3 = 0
\end{array}$$

Because the constants c_i are defined up to a translation, we can add the constraints $\gamma_1 \ge 0$ and $\gamma_2 \ge 0$ in order to obtain positive execution times. We would have found for the matrix-matrix product: $c_1 = 0$, $c_2 = 0$ et $c_3 = n$.

6.2 Interpretation of the affine-by-statement schedule

For one statement, we have seen that the duality theory can give an interpretation of the execution time of the best linear schedule as the length of the longest dependence path, that is to say the time of the free schedule. Let us look how to interpret the time of the optimal affine-by-statement schedule.

Denote by E the set of all couples of vertices (i, j) that correspond to edges of the oriented reduced graph describing the system of equations. By the duality theorem of linear programming ([20, p.90]), the linear problem obtained above is the dual of the following one:

$$\left\{\begin{array}{l}
\text{Fractions of path along self-dependences:}\\
\forall i, 1 \le i \le k\\
Y_i \ge 0\\
D_iY_i + \sum_{j\in Succ(i)} X_{i,j}d_{i,j} = p_i - q_i + \sum_{j\in Succ(i)} r_{i,j} - \sum_{j\in Pred(i)} r_{j,i}\\
\\
\text{Intermediate points:}\\
Ap_i \le \lambda_i b\\
Aq_i \le \mu_i b\\
Ar_{j,i} \le X_{j,i}b\\
\\
\text{Eulerian path with components } X_{i,j} \text{ on each edge}\\
X_{i,j} \ge 0\\
\sum_{j\in Pred(i)} X_{j,i} - \sum_{j\in Succ(i)} X_{i,j} = \lambda_i - \mu_i\\
\\
\text{Barycentric condition on path repartition:}\\
\sum_{1\le i\le k} \lambda_i = 1 \text{ et } \lambda_i \ge 0\\
\sum_{1\le i\le k} \mu_i = 1 \text{ et } \mu_i \ge 0\\
\\
\max(\sum_{1\le i\le k} \|Y_i\|_1 + \sum_{(j,i)\in E} X_{j,i})
\end{array}\right.$$

($\|X\|_1$ represents the norm 1 of X, i.e. the sum of its components.) The solution of this problem corresponds to a "rational path" going $X_{i,j}$ times along edge (i,j) and Y_i times along the self-dependences. Indeed, if we add the k first equations, we obtain:

$$\sum_{1\le i\le k} D_iY_i + \sum_{(i,j)\in E} X_{i,j}d_{i,j} = p - q + \sum_{(i,j)\in E} r_{i,j} - \sum_{(i,j)\in E} r_{i,j} = p - q$$

where $p = \sum_{1\le i\le k} p_i$ satisfies $Ap \le b$ (idem for q). The objective function is exactly the length of this path. However, how can we reconstruct, from these data, an actual path of dependences, that is to say a path going along edges with an order corresponding to a feasible path in the reduced graph and a path staying at any time in the domain $\{x, Ax \le b\}$? Consider the simple case of a fork graph ($(k-1)$ roots and one leaf), as for the matrix-matrix product for example, and let us examine the solutions of this problem which satisfy that the λ_i's and the μ_i's are integers. There are two cases: either there is an i such that $\lambda_i = \mu_i = 1$ and the situation is similar to the case of one statement, or $\lambda_k = 1$ and $\mu_0 = 1$ by renumbering the equations. We see that, because the $X_{i,j}$ are positive, this is possible only if 0 is the index of a root and k is the index of the leaf. Noting that $Ap \le 0$ implies $p = 0$ if the domain has no ray, we see that a solution of this problem with integer λ_i's and μ_i satisfies:

$$\begin{cases} \forall i, i \neq 1, i \neq k \\ Y_i = p_i = r_{i,k} = X_{i,k} = 0 \\ \\ D_0 Y_0 + X_{0,k} d_{0,k} = r_{0,k} - q_0 \\ D_k Y_k = p_k - r_{0,k} \\ \\ X_{0,k} = 1 \\ A p_k \leq b \\ A q_0 \leq b \\ A r_{0,k} \leq b \\ \\ \max(\|Y_1\|_1 + \|Y_2\|_1 + \sum_{(j,i)\in E} X_{j,i}) \end{cases}$$

With a similar rouding technique as the one used for one statement (cf [2]), we can reconstruct an actual path of same length up to a constant, going from q_0 to $r_{0,k}$ along self-dependence vectors of statement 0, then from $r_{0,k}$ to p_k along self-dependence vectors of statement k. This remains true for a directed tree of any heigth.

6.3 The affine-by-statement schedule is not optimal

In the general case, the λ_i's and the μ_i's are not necessary integers. The path can be divided into a fraction of path coming from statement 1 and a fraction of path coming from statement 2. If we examine where the λ_i's and the μ_i's are coming from in the duality, we see that there are coming as the dual variables corresponding to the computation of the last time step (constraints on γ_1) and of the first time step (constraints on γ_2). By the theorem of complementary slackness (see [20, p.95]), if there is a solution for which this maximum (γ_1) (respectively this minimum (γ_2)) is reached for only one statement then all λ_i's (respectively μ_i's) are zero except one. In this case, the schedule is nearly optimal as we have briefly said above. The non-optimality comes from a situation where for example two roots start computing at the same time and stop at the same time. We can try to solve the general linear problem and then if γ_2 is reached for two statements, we solve the problem again but without the constraint on γ_2 and one of the statements in order to force one of the μ_i to be 0. If the minimum does not decrease, the optimal solution gives variables for which the dual variables are zero for one more μ_i. Going this way, one can check the nearly-optimality of the solution. This is what happens for the matrix-matrix product. If we remove the condition $\gamma_2 - X_1'' b + c_1 \geq 1$, we obtain the same minimum value for which the inequalities on γ_1 (respectively γ_2) are all strict except one. This implies that in the dual problem, there is a solution where all the λ_i's (respectively μ_i's) are zero except one.

However, if the statements are very dependent (i.e. if the minimum decreases), we might have the non-optimality. Considering the dual problem permits to give a lower bound on the best affine-by-statement schedule by taking the value of any solution in the dual problem. This gives us the counter-example for the nearly-optimality of this class of schedules. On a reduced graph like the one of matrix-matrix product, let us look at the solutions of the form:

$$\left\{\begin{array}{l} Y_1 \geq 0 \\ Y_2 \geq 0 \\ Y_3 \geq 0 \\ \\ D_1 Y_1 = r_{1,3} - q_1 \\ D_2 Y_2 = r_{2,3} - q_2 \\ D_3 Y_3 = p_3 - \frac{1}{2}(r_{1,3} + r_{2,3}) \\ \\ A p_3 \leq b \\ A q_1 \leq b \\ A q_2 \leq b \\ A r_{1,3} \leq b \\ A r_{2,3} \leq b \\ \\ X_{1,3} = \frac{1}{2} \\ X_{2,3} = \frac{1}{2} \\ \\ \max(\frac{1}{2}\|Y_1\|_1 + \frac{1}{2}\|Y_2\|_1 + \|Y_3\|_1 + 1) \end{array}\right.$$

which corresponds to the case $\lambda_3 = 1$, $\mu_1 = \mu_2 = \frac{1}{2}$ after having multiplied by 2 the equations on statements 1 et 2. If $q_1 = q_2$, the solution corresponds to half the length of a fraction of path from q_1 to $r_{1,3}$, plus half the length of a fraction of path from q_1 to $r_{2,3}$, plus the length of a fraction of path from the middle of $r_{1,3}$ and $r_{2,3}$ to p_3. This does not correspond to an actual dependence path but to a repartition on both statements 1 and 2. For the example given on figure 6, there is a fictitious path of length $2n$ but the length of the longest path is $\frac{3}{2}n$. Here, the dependence vectors are:

$$\begin{array}{lll} d_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} & d_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} & d_3 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \\ d_{1,3} = 0 & d_{2,3} = 0 & \end{array}$$

Thus, the affine-by-statement schedule is not optimal. In this example, the dependence vectors are not lexicographically positive. One can also build a counter-example corresponding to a loop nest. Consider a n-by-n square domain ($0 \leq x, y \leq n$), with again the same reduced graph as for matrix-matrix product but with the following dependence vectors:

$$\begin{array}{lll} d_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} & d_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} & d_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ d_{1,3} = d_1 & d_{2,3} = d_2 & \end{array}$$

the time of the best affine-by-statement schedule is $\frac{3}{2}n$ with the solution:

$$\begin{array}{lll} X_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} & X_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} & X_3 = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \\ c_1 = 0 & c_2 = 0 & c_3 = n/2 \end{array}$$

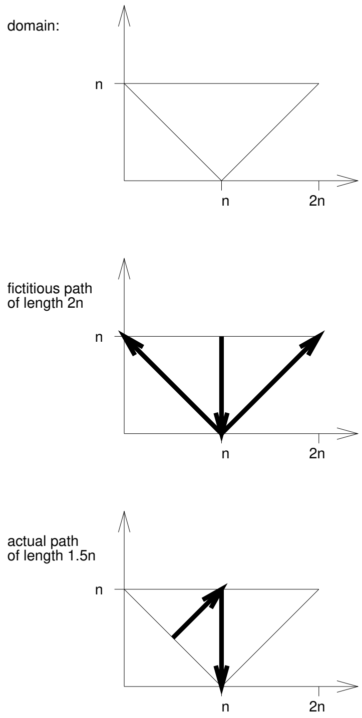

Figure 6: Counter-example for the optimality of affine-by-statement schedules

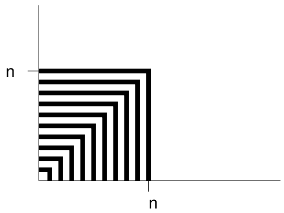

Figure 7: The optimal schedule for the second counter-example

whereas the length of the longest path is n. Actually, one can schedule in time n the statements S_3 as shown figure 7. Thus, the execution time of the optimal affine-by-statement schedule is here greater of 50% than the execution time of the free schedule.

7 Conclusion

We have surveyed the main techniques for scheduling uniform loop nests. From a theoretical viewpoint, we list below our prominent results:

- for a single statement (or for several statements processed as a single block), the optimal linear schedule is asymptotically equivalent to the free schedule.
- this result does not generalize to the case of several statements: the affine-by-statement schedule is not always nearly optimal. This negative result should

not be over-estimated, since affine-by-statement scheduling is always better than linear scheduling - and the difference is often significant -, while it is not more complex from an implementation viewpoint.

- introducing a different constant for each statement (while keeping the same scheduling vector) is a simple but powerful technique that permits to retrieve and to improve some known results on cycle shrinking.

From a practical viewpoint, we have proposed a new, constructive and efficient method for determining the optimal parameters of the three scheduling classes (linear, linear + constant, affine-by-statement). This method relies upon a constraint imposed by the shape of the computation domain that must be satisfied by any admissible scheduling vector.

Needless to say, the scheduling of uniform loop nests represents a small step toward the design of a powerful compiler-parallelizer ... but we believe it is a *sine-qua-non* for such an exciting enterprise !

8 Annex

Rajopadhye [16] and Quinton [11] have proposed to extend the uniform dependence algorithms to affine dependences called ARE (Affine Recurrences Equations). The dependences, instead of linking the point p to the $p - d_{i,j}$, link each point p of the domain to the point $I_{i,j}(p)$ where $I_{i,j}(p) = D_{i,j}p - d_{i,j}$ is an affine function of the indices. The technique for searching the best scheduling vector can be generalized to this case, by replacing, as we have seen, a set of constraints depending on the points of the domain by two matrix equations. For instance, in the case of a single equation and k self-dependences $D_i p - d_i$, the constraint $XD \geq 1$ become:

$$\begin{cases} \forall p, Ap \leq b \\ \forall i, 1 \leq i \leq k \\ Xp - X(D_i p - d_i) \geq 1 \end{cases}$$

and is replaced by:

$$\begin{cases} \forall i, 1 \leq i \leq k \\ X(I - D_i) = -X_i A \\ X_i \geq 0 \\ -X_i b + X d_i \geq 0 \end{cases}$$

In the same way, for a system of equations, one can replace the constraints $X_j - X_i = X_{j,i}A$ by $X_j D_{j,i} - X_i = X_{j,i}A$.

Consider the following example which correspond to the algebraic path problem (see [18]):

for $k = 1$ to n do
  for $i = k$ to $k + n$ do
  for $j = k$ to $k + n$ do
    $a(i, j, k) = a(i, j, k - 1) + a(i, k, k - 1).a(k, k, k - 1)^*.a(k, j, k - 1)$

In this simple example, with one instruction, we find the dependences:

$$D_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad D_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad D_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad D_4 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$d_1 = d_2 = d_3 = d_4 =^t (1, 0, 0)$$

Solving the associated linear problem leads to the natural solution:

$$X =^t (1, 0, 0)$$

Acknowledgments

This work has been supported by the Project C3 of the French Council for Research CNRS, and by the ESPRIT Basic Research Action 3280 "NANA" of the European Economic Community.

It is pleasure to thank Patrice Quinton for several discussions and working sessions.

References

[1] U. Banerjee. An introduction to a formal theory of dependence analysis. *The Journal of Supercomputing*, 2:133–149, 1988.

[2] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. Technical Report 91-35, Laboratoire de L'informatique du Parallélisme, Ecole Normale Supérieure de Lyon, November 1991. to appear in Parallel Processing Letters.

[3] Michael Dowling. Optimal code parallelization using unimodular transformations. In *Parallel Computing*, volume 16, pages 157–171. Elsevier Science Publishers B.V., North Holland, 1990.

[4] Paul Feautrier and Nadia Tawbi. Résolution de systèmes d'inéquations linéaires; mode d'emploi du logiciel pip. Technical Report 90-2, Institut Blaise Pascal, UPMC, Laboratoire MASI, January 1990.

[5] Seema Hiranandani, Ken Kennedy, and Chau-Wen Tseng. Compiler optimizations for fortran d on mimd distributed-memory machines. In *Supercomputing 91*, pages 86–100. IEEE Computer Society Press, November 1991.

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

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

[8] C.E. Leiserson and J.B. Saxe. Optimizing synchronous systems. *Journal of VLSI Computers Systems*, 1:41–67, April 1983.

[9] Bjorn Lisper. Linear programming methods for minimizing execution time of indexed computations. In P. Feautrier and F. Irigoin, editors, *International Worshop on Compilers for Parallel Computers*, pages 131–142, MASI, PARIS, December 1990.

[10] L.S. Liu, C.W. Ho, and J.P. Sheu. On the parallelism of nested for-loops using index shift method. In *Proceedings of International Conference on Parallel Processing*, volume 2, pages 119–123, August 1990.

[11] Christophe Mauras, Patrice Quinton, Sanjay Rajopadhye, and Yannick Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In S.Y Kung, Jr. E.E. Swartzlander, J.A.B. Fortes, and K.W. Przytula, editors, *Application Specific Array Processors*, pages 100–110. IEEE Computer Society Press, September 1990.

[12] J.K. Peir and R. Cytron. Minimum distance: a method for partitioning recurrences for multiprocessors. *IEEE Transactions on Computers*, 38(8):1203–1211, August 1989.

[13] C.D. Polychronopoulos. Compiler optimization for enhancing parallelism and their impact on architecture design. *IEEE Transactions on Computers*, 37(8):991–1004, August 1988.

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

[15] Patrice Quinton. The systematic design of systolic arrays. In Françoise Fogelman Soulie, Yves Robert, and Maurice Tchuente, editors, *Automata Networks in Computer Science*, chapter 9, pages 229–260. Manchester University Press, 1987.

[16] Sanjay V. Rajopadhye and Richard M. Fujimoto. Synthesizing systolic arrays from recurrence equations. In *Parallel Computing*, volume 14, pages 163–189. Elsevier Science Publishers B.V., North Holland, 1990.

[17] Sailesh K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, October 1885.

[18] Tanguy Risset and Yves Robert. Synthesis of processor arrays for the algebraic path problem. *Parallel Processing Letters*, 1(1):19–28, September 1991.

[19] Yves Robert and Siang W. Song. Revisiting cycle shrinking. Technical Report 91-31, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, September 1991. to appear in Parallel Computing.

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

[21] Weijia Shang and A.B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Transactions on Computers*, 40(6):723–742, June 1991.

[22] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* MIT Press, Cambridge MA, 1989.