Scheduling the Computations of a Loop Nest with Respect to a Given Mapping

Alain Darte[1], Claude Diderich[2], Marc Gengler[3], and Frédéric Vivien[4]

[1] LIP, École normale supérieure de Lyon, F-69364 Lyon, France.
[2] Wannerstrasse 21, CH-8045 Zurich, Switzerland.
[3] LIM, École Supérieure d'Ingénierie de Luminy, F-13288 Marseille cedex 9, France.
[4] ICPS, Université Louis Pasteur, Strasbourg, Pôle Api, F-67400 Illkirch, France.

Abstract. When parallelizing loop nests for distributed memory parallel computers, we have to specify when the different computations are carried out (computation scheduling), where they are carried out (computation mapping), and where the data are stored (data mapping). We show that even the "best" scheduling and mapping functions can lead to a sequential execution when combined, if they are independently chosen. We characterize when combined scheduling and mapping functions actually lead to a parallel execution. We present an algorithm which computes a scheduling compatible with a given computation mapping, if such a schedule exists.

1 Introduction

When parallelizing codes for distributed memory parallel computers, it is fundamental to develop efficient strategies to distribute the workload between processors, and to distribute the data involved by these computations. Indeed, for such machines, communications between processors and global synchronizations are very expensive compared to the computation speed of the processors. The problem is to find an acceptable trade-off between the two extreme solutions, a one processor execution that involves no external communication, but sequentializes all computations, and the maximal distribution of computations that exploits all parallelism but whose performance may be damaged by too many communications or synchronizations. In the field of automatic parallelization of nested loops, this problem has been cut into two sub-problems known as the *mapping problem* and the *scheduling problem*.

The first problem is the mapping, to the different processors, of the computations (i.e., the loop iterations) and of the data elements involved by them. This mapping is usually done as follows: a first step (the *alignment* phase) defines a mapping on a δ-dimensional grid of *virtual* processors, the goal being to minimize the amount of communication overhead due to non local data references. The dimension δ is usually an input to this problem. Then, a second step (the *distribution* phase) defines a mapping of the virtual processors onto physical processors. This two-step scheme follows the same principle as in HPF. The *alignment* phase can be viewed as a way to automatically derive HPF *align* directives. Different formulations of the mapping problem were studied. The mapping has been studied, in similar linear algebra frameworks, among others, by Ramanujam and Sadayappan [11], Anderson and Lam [2], Bau *et al.* [3], Dion and Robert [7], Feautrier [9], and Diderich and Gengler [6].

The second problem is the definition of a partial order for the execution of the loop iterations. This order must respect the dependences in the code. It is used to rewrite the code so as to make explicit the sequential steps (more or less the global synchronizations) required by the semantics of the code. In the context of HPF, scheduling can be viewed as a way to automatically detect *independent* directives. The main algorithms, using exact representations of data dependences, are those of Feautrier [8], and Lim and Lam [10]. Allen and Kennedy [1], Wolf and Lam [12], and Darte and Vivien [5] introduced parallelism detection and scheduling algorithms that use a conservative approximation of the data dependences.

Until now, both problems - the mapping and the scheduling problems - have generally been studied separately. In most works on scheduling, the mapping is supposed to fit well with the scheduling. However, there is no reason for a given scheduling to lead to an efficient execution, if communication costs are not taken into account. It is possible that the scheduling enforces some computations to be executed by different processors, and that this “inherent” mapping involves very expensive communications. In most works on mapping, the scheduling problem is not addressed at all: the code is supposed to be compiled, for example as an HPF program, following the owner computes rule (the processor that performs an assignment is the processor that owns the memory cell that is assigned). In the least favorable case, this may lead to very poor performances, since the order in which computations are carried out is not optimized. There is indeed no reason for a particular alignment to lead to a parallel execution of the computations, if the scheduling problem is not taken into account. It is very possible that the mapping enforces a sequential utilization of the processors when respecting the data dependences in the code.

This paper is a first step in the direction of a simultaneous solution to both the mapping and the scheduling problems. We illustrate, in Section 2, why both problems cannot be solved independently in general. Then, in Section 3, we formally state the problem of compatibility between mapping and scheduling functions. In Section 4, we present our solution on an example. In Section 5, we characterize the mappings for which there exists a compatible scheduling. Finally, in Section 6, we describe an algorithm that effectively builds a compatible scheduling for a given mapping, when one exists. We conclude with some perspectives and extensions of our results.

Note: the missing proofs and explanations can be found in [4].

2 Compatibility of Mapping and Scheduling Functions

We consider here the (uniform) loop nest presented as Example 1. Suppose that we are looking for a one-dimensional alignment of this loop nest, that is, we consider the processors to be indexed as a vector of processors. As usually, we search for an alignment which minimizes the cost of non local memory accesses. If communicated data are not kept in memory for multiple reuse the optimal 1D-alignment maps the operations `S(I,*)` and the data elements `A(I-1,*)` to processor `P(I)` which yields three local accesses `A(I-1,J-1)`, `A(I-1,J)`, and `A(I-1,J+1)`, and one remote to store `A(I,J)` (thereby breaking the owner computes rule). This alignment is not compatible with the implicit (and shortest) scheduling given by the `DOSEQ-DOALL` form: processor `P(I)` would compute all computations `S(I,*)` at time-step `I` and would thus serialize them. Nevertheless, one can find schedules compatible with the given alignment: any schedule of the form $a\mathtt{I} + b\mathtt{J}$, with $a > b > 0$, is compatible and valid, like the function which schedules `S(I,J)` at time-step $2 * \mathtt{I} + \mathtt{J}$. In terms of program transformation, this schedule is equivalent to a loop skewing and a loop interchange. In this example, the linear part of the computation mapping is given by the vector $(1, 0)$ (i.e., `(I,J)` is mapped onto `P(I)`) and the linear part of the scheduling by the vector $(2, 1)$. The compatibility can be read from the fact that $(2, 1)$ and $(1, 0)$ are linearly independent. However, if the scheduling `DOSEQ-DOALL` is imposed, we have to change the alignment. One possible solution is to map `A(I,J)` on processor `P(J)` (thus with two non local accesses, instead of one). In this example, either the scheduling or the alignment can be chosen optimal, but the optimal scheduling and the optimal alignment are incompatible. There exist of course cases where the optimal scheduling and alignment are compatible.

Example 1.
```
DOSEQ I = 1, N
 DOALL J = 1, N
S A(I,J) = A(I-1,J-1) + A(I-1,J) + A(I-1,J+1)
 ENDDO
ENDDO
```

3 Statement of the Problem

The problems of mapping and scheduling were both mainly studied in the affine framework. In this framework, the mapping and scheduling functions are (multi-dimensional) affine functions, and the virtual processors form a grid. We suppose that we want to parallelize a loop nest while exhibiting δ degrees of parallelism. The scheduling functions must be such that the δ dimensions of the mapping are actually parallel: at each time step defined by the scheduling (in steady state) some operations are executable independently; we want the mapping to project this set of computations onto a set of processors of dimension δ. A schedule and a computation mapping which satisfy this property are said *compatible*.

As illustrated by Example 1, an alignment that minimizes the communication and a scheduling that expresses the maximum achievable parallelism can lead to a completely sequential execution when used together. We thus have to consider both problems simultaneously, or at least to solve one with respect to the other. A general approach consists in computing a "good" alignment that is compatible with at least one possible scheduling. Indeed, the constraints on the scheduling are mandatory, while the constraints on the locality of the accesses are not. If an alignment constraint cannot be met, this means that one access will be remote and will slow down the execution speed but will not affect correctness. So, we start by computing an optimal (optimal with respect to some communication cost) alignment and we check whether there is a scheduling compatible with it. If so, we keep both the alignment and the scheduling. On the contrary, we look for the next best alignment, checking whether it is compatible with some scheduling function, and so on. We thus have to characterize what we mean by *compatible*, to characterize mappings for which there is at least one compatible scheduling, and to provide an algorithm that builds such a scheduling when it exists.

3.1 Hypotheses and Notations

We assume that the alignment problem has been solved and we make no assumption on the mapping we are given. We focus on the scheduling problem of a single loop nest that we assume to be perfectly nested, of depth n, and containing s assignment instructions. For each instruction S, the *scheduling* function assigns a (multi-dimensional) execution date to each loop iteration and is written

$$\begin{aligned} E_S \colon \ \mathcal{D} &\longrightarrow \mathbb{T}_S \\ \boldsymbol{i} &\longmapsto E_S(\boldsymbol{i}) = \mathbf{E}_S\, \boldsymbol{i} + \boldsymbol{e}_S \end{aligned}$$

where $\mathbb{T}_S$ is the d_S-dimensional time space associated with S (it is a subset of $\mathbb{Z}^{d_S}$). $E_S(\boldsymbol{i})$ is the time-step when iteration $\boldsymbol{i}$ of instruction S is scheduled (time-steps are lexicographically ordered). Similarly, for each instruction S, the *mapping* function assigns a processor to each loop iteration and is written

$$\begin{aligned} C_S \colon \ \mathcal{D} &\longrightarrow \mathbb{P} \\ \boldsymbol{i} &\longmapsto C_S(\boldsymbol{i}) = \mathbf{C}_S\, \boldsymbol{i} + \boldsymbol{c}_S \end{aligned}$$

where $\mathbb{P}$ is the virtual δ-dimensional grid of processors. $C_S(\boldsymbol{i})$ is the processor on which iteration $\boldsymbol{i}$ of instruction S is executed. All matrices $\mathbf{C}_S$ and $\mathbf{E}_S$ are assumed to be of full row rank.

There exist different equivalent criteria to define compatibility. We can say that the scheduling function E_S and the mapping function C_S of an instruction S are *compatible* if and only if at any time any virtual processor is supposed to execute a limited number of iterations that does not depend on the loop bounds (that may be parameterized). Mathematically, this is equivalent to:

$$\operatorname{rank} \begin{pmatrix} \mathbf{E}_S \\ \mathbf{C}_S \end{pmatrix} = d_S + \delta. \tag{1}$$

Indeed, if this rank is not $d_S + \delta$, there is a nonzero vector $\boldsymbol{x}$ such that $\mathbf{E}_S\, \boldsymbol{x} = 0$ and $\mathbf{C}_S\, \boldsymbol{x} = 0$. Consequently, all iterations $\boldsymbol{i}' = \boldsymbol{i} + \lambda \boldsymbol{x}$ are performed at the same time $E_S(\boldsymbol{i})$ and on the same virtual processor $C_S(\boldsymbol{i})$, whatever the integer λ. This matrix constraint is well known when applying loop transformations. Here the first dimensions correspond to time, the last dimensions to space.

3.2 The Underlying Scheduler

In this paper, we solve our problem for a particular scheduling algorithm called Darte-Vivien and fully detailed in [5]. This algorithm generalizes both the Allen-Kennedy algorithm [1] and the Wolf-Lam algorithm [12]. It works on an over-approximation of dependences by polyhedra. Here, we only state the details of Darte-Vivien needed to understand the rest of this paper. This algorithm produces, for each statement S, a multidimensional affine function: $(S, \boldsymbol{i}) \mapsto (\mathbf{E}_S^1 \boldsymbol{i} + \rho_S^1, \dots, \mathbf{E}_S^{d_S} \boldsymbol{i} + \rho_S^{d_S})$, where d_S denotes the dimension of the schedule for S. Different statements may have different schedule dimensions. Briefly speaking, Darte-Vivien computes recursively some strongly connected subgraphs denoted $G_u(S,i)$. The graphs $G_u(S,i)$ contain some nodes which correspond to statements and which are called *actual*, and some other nodes which are called *virtual*. If $i \leq d_S$, the graph $G_u(S,i)$ is defined as the strongly connected component, containing S, of the subgraph $(G_u(S,i-1))'$ of $G_u(S,i-1)$ generated by all the edges that can not be satisfied by the first $(i-1)$ dimensions of any schedule. If a statement T is in $G_u(S,i)$, then $G_u(S,i) = G_u(T,i)$ and statements S and T have the same i-th linear part $\mathbf{E}_S^i$ in their schedules. If C is a set of edges of $G_u(S,i)$, $\boldsymbol{w}(C)$ is the sum of the dependence weights along C, and $l(C)$ is the number of edges in C whose tail is an actual node and which are satisfied by the i-th dimension of the schedule. Then the linear part of the schedule of any statement S is easily characterized: the set of admissible $\mathbf{E}_S^i$ is the polyhedron $\mathsf{P}(S,i)$: $\{X \mid \forall \mathcal{C} \text{ cycle of } G_u(S,i), X\boldsymbol{w}(\mathcal{C}) \geq l(\mathcal{C})\}$. Conversely, any collection of vectors $\mathbf{E}_S^i \in \mathsf{P}(S,i)$ such that $\mathbf{E}_S^i = \mathbf{E}_T^i$ for each statement T in $G_u(S,i)$ is the valid linear part of a schedule. Thus, we can explicit the set of all possible schedules (up to some regularity conditions). In this set we will look for one schedule compatible with the given mapping. Finally, let $\mathsf{VS}(S,i)$ be the vector space generated by $\mathsf{P}(S,i)$ ($\mathsf{VS}(S,i) \subsetneq \mathsf{VS}(S,i+1)$).

4 Example

In this section, we illustrate on an example how to build a schedule compatible with a given computation mapping. Here we chose a simple example for clarity. It illustrates the main lines of our technique but does not exhibit all the complexity of the problem, which appears only for some loop nests of dimension at least 3. The existence condition of a compatible schedule is presented in Section 5. The formal algorithms used to build such a schedule are presented in Section 6.

The Mappings. Figure 1 presents Example 2 and its (uniform) dependences. Here we look for a one-dimensional schedule and a one-dimensional mapping. We assume that, possibly because of other loop nests, data `a(I,J)` and operation S_1`(I,J)` are mapped onto processor `I` (the linear part of the mapping is then vector $C_{S_1} = (1,0)$), and data `b(I,J)`, data `c(I,J)`, and operation S_2`(I,J)` are mapped onto processor `J` (the linear part of the mapping is then vector $C_{S_2} = (0,1)$). The linear part of the schedule must be linearly independent of the mapping directions (Condition (1)). As the mapping functions are $(0,1)$ and $(1,0)$, we cannot use a schedule whose linear part is parallel to one of the axes.

Example 2.

```
DO I=1,N
 DO J=1,N
  S1 a(I,J) = b(I-1,J-1)+c(J,I)
  S2 b(I,J) = a(I-1,J)+a(I,J-1)+c(I,J)
 ENDDO
ENDDO
```

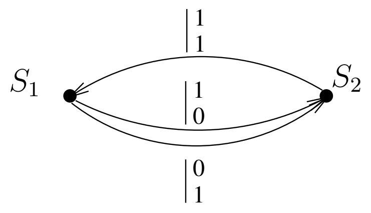

Fig. 1. Code and dependence graph for Example 2.

Constraints on the Schedule. Let vector X and constants ρ_{S_1} and ρ_{S_2} define a one-dimensional affine schedule: S_k`(I,J)` is then scheduled at time $\mathtt{X(I,J)}^t + \rho_{S_k}$, $k \in \{1,2\}$. The three dependences give three constraints on this schedule:

– S_1`(I,J)` depends on S_2`(I-1,J-1)`. Therefore, $(X(\mathtt{I},\mathtt{J})^t + \rho_{S_1})$ must be greater than $1 + (X(\mathtt{I}-1,\mathtt{J}-1)^t + \rho_{S_2})$, i.e., $X(1,1)^t + \rho_{S_1} - \rho_{S_2} \geq 1$.
– S_2`(I,J)` depends on S_1`(I-1,J)`. Therefore, $X(1,0)^t + \rho_{S_2} - \rho_{S_1} \geq 1$.
– S_2`(I,J)` depends on S_1`(I,J-1)`. Therefore, $X(0,1)^t + \rho_{S_2} - \rho_{S_1} \geq 1$.

From the previous set of constraints, we infer that the vector $X = (x,y)$ is the linear part of a valid one-dimensional schedule if and only if it belongs to the polyhedron: $\mathsf{P} = \{(x,y) \,|\, 2x + y \geq 2 \text{ and } x + 2y \geq 2\}$. This polyhedron generates the vector space $\mathsf{VS} = \mathbb{Q}^2$.

The Scheme. A schedule compatible with the mapping is built in three steps:

1. We build a vector $\mathbf{F} \in \mathsf{VS}$ satisfying Equation (1) for both S_1 and S_2 ($\mathsf{VS} \supset \mathsf{P}$).
2. From $\mathbf{F}$, we build a vector $\mathbf{E} \in \mathsf{P}$ satisfying Equation (1) for both S_1 and S_2.
3. We compute the constants ρ_S that, associated with $\mathbf{E}$, define a valid schedule.

Building a Solution in the Vector Space. We need a vector in the vector space $\mathsf{VS} = \mathbb{Q}^2$ which belongs neither to $\mathsf{C}(S_1) = \text{Span}\{(1,0)\}$ nor to $\mathsf{C}(S_2) = \text{Span}\{(0,1)\}$. We consider a vector in VS, but not in $\mathsf{C}(S_1)$ (resp. $\mathsf{C}(S_2)$), say $X_1 = (0,1)$ (resp. $X_2 = (1,0)$). Neither of them is a solution as $X_1 \in \mathsf{C}(S_2)$ and $X_2 \in \mathsf{C}(S_1)$. But any other vector on the line defined by X_1 and X_2 is independent with both C_{S_1} and C_{S_2}, e.g. $(X_1 + X_2)/2 = (1/2, 1/2)$. To get an integral solution, we scale this vector and we obtain: $\mathbf{F} = (1,1)$.

Building a Solution in the Polyhedron. We know a vector $\mathbf{F}$ in the vector space $\mathsf{VS} = \mathbb{Q}^2$ which is linearly independent with both the vectors C_{S_1} and C_{S_2}. What we need is a vector $\mathbf{E}$ of P with the same property. In fact $(1,1)$ belongs to P and our problem is already solved! To show how to proceed when we are not so lucky, suppose we found the vector $(1,-1)$ of VS, which also belongs neither to $\mathsf{C}(S_1)$ nor to $\mathsf{C}(S_2)$. First we consider an arbitrary vector $\mathbf{P}$ of P, e.g. $\mathbf{P} = (1,1)$ (such a vector can easily be found by linear programming [5]). We want to add λ times the vector $\mathbf{P}$ to $\mathbf{F}$ so as to obtain a vector $\mathbf{E} = \mathbf{F} + \lambda\mathbf{P}$ which belongs to P and is linearly independent with the vectors C_{S_1} and C_{S_2}. As $\mathsf{P} = \{(x,y) \mid 2x + y \geq 2 \text{ and } x + 2y \geq 2\}$, condition $(\mathbf{F} + \lambda\mathbf{P}) \in \mathsf{P}$ is equivalent to $\lambda \geq 1$. We cannot choose $\lambda = 1$ which leads to $\mathbf{E} = (2,0)$ which is collinear with C_{S_1}. We can take $\lambda = 2$ which gives the solution $\mathbf{E} = (3, 1)$. Note that this mechanism gives in general *a* solution, not an *optimal* solution.

Computing the Constants. We have built the linear part of our schedule, say $\mathbf{E} = (1, 1)$, but we still need the constants. The constants can be computed using a shortest-path algorithm. In our example, this is not needed: the inner product of $(1, 1)$ with each distance vector is already greater than 1, so we can take $\rho_{S_1} = \rho_{S_2} = 0$. S_1`(I,J)` and S_2`(I,J)` are both computed at time `I+J`. At time `T`, processor `P` only has to execute the two operations S_1`(P,T-P)` and S_2`(T-P,P)`. Here is the code corresponding to the whole transformation:

```
DOSEQ T = 2, 2N
 DOALL P = max(T-N,1), min(N,T-1)
  S1 a(P,T-P) = b(P-1,T-P-1)+c(T-P,P)            /* on processor P */
  S2 b(T-P,P) = a(T-P-1,P)+a(T-P,P-1)+c(T-P,P)   /* on processor P */
 ENDDO
ENDDO
```

5 Existence of a Compatible Schedule

As stated in Section 3, we need to find, for each statement S and each integer i in $[1, d_S]$, a vector $\mathbf{E}_S^i$ in $\mathsf{P}(S, i)$ such that the vectors $\mathbf{E}_S^1, ..., \mathbf{E}_S^{d_S}, \mathbf{C}_S^1, ..., \mathbf{C}_S^\delta$ are linearly independent and such that $\mathbf{E}_S^i = \mathbf{E}_T^i$ for each statement T in $G_u(S, i)$.

Lemma 1 (Existence of a Solution for Darte-Vivien).
Let $\mathsf{C}(S)$ denote the vector space generated by the vectors $\{\mathbf{C}_S^1, ..., \mathbf{C}_S^\delta\}$. We can associate to each statement S a sequence of vectors $\mathbf{E}_S^1, ..., \mathbf{E}_S^{d_S}$ such that:

1. *$\mathbf{E}_S^i \in \mathsf{P}(S, i)$;*
2. *all the statements T of $G_u(S, i)$ have the same i-th vector $\mathbf{E}_S^i$;*
3. *the vectors $\mathbf{E}_S^1, ..., \mathbf{E}_S^{d_S}, \mathbf{C}_S^1, ..., \mathbf{C}_S^\delta$ are linearly independent;*

if and only if, for each statement S, each integer i in $[1, d_S]$,

$$i + \dim(\mathsf{VS}(S, i) \cap \mathsf{C}(S)) \leq \dim(\mathsf{VS}(S, i)) \tag{2}$$

This lemma gives a necessary and sufficient condition for the existence of a schedule compatible with a given computation mapping, the underlying scheduling algorithm being Darte-Vivien. This condition states the existence of a compatible schedule iff there is one among those that Darte-Vivien can build. One could wonder whether there are examples for which there exist affine schedules compatible with the given computation mapping, but no such schedules among those Darte-Vivien can build. In fact, this cannot occur when dependence distances are approximated by polyhedra [4]. Condition (2) is a general condition.

6 The Algorithm

The algorithm, which builds the desired schedule when Condition (2) of Lemma 1 is fulfilled, proceeds in three steps: 1) building of the vectors $\mathbf{F}_S^i \in \mathsf{VS}(S,i)$ satisfying the desired properties; 2) from the vectors $\mathbf{F}_S^i$, building of the vectors $\mathbf{E}_S^i \in \mathsf{P}(S,i)$ satisfying the desired properties; 3) computing the constants ρ_S^i that, associated with the vectors $\mathbf{E}_S^i$, define a valid schedule.

6.1 Construction of the Vectors

In the algorithms listed below, each vector space is defined by one of its basis.

– Algorithm BUILD_VECTORS takes as inputs the vector spaces $\mathsf{VS}(S,i)$ and $\mathsf{C}(S)$, and builds the desired $\mathbf{F}_S^i \in \mathsf{VS}(S,i)$ iff Condition (2) is fulfilled.

```
BUILD_VECTORS
For $i = 1$ to $\max_{S\in G_u} d_S$ do
  For each subgraph $G_u(S,i)$ do
    Let $T_1, ..., T_p$ be the $p$ statements in $G_u(S,i)$.
    $x =$ IN&OUT(Span($\mathbf{F}_S^1, ..., \mathbf{F}_S^{i-1}$)+$\mathsf{C}(T_1)$, ..., Span($\mathbf{F}_S^1, ..., \mathbf{F}_S^{i-1}$)+$\mathsf{C}(T_p)$;
                $\mathsf{VS}(S,i)$)).
    For each $T$ in $G_u(S,i)$ let $\mathbf{F}_T^i = x$.
```

– Algorithm IN&OUT takes as input some subspaces of $\mathbb{Q}^n$, $F_1, ..., F_m$, and E. It outputs a vector $x \in (E \setminus \cup_{j=1}^m F_j)$.

```
IN&OUT($F_1, ..., F_m; E$)
$x_1 =$ FIND_POINT_NOT_IN($F_1, E$).
For $i = 2$ to $m$ do
  $y =$ FIND_POINT_NOT_IN($F_i, E$).
  $H = \{\lambda x_{i-1} + (1-\lambda)y \mid \lambda \in \{0, \frac{1}{i}, ..., \frac{i}{i}\}\}$
  Choose $x_i$ in $H$ such that: $\forall j \in [1,i]$, POINT_IS_NOT_IN($x_i, F_j$) = TRUE.
Return $x_m$.
```

– Algorithm FIND_POINT_NOT_IN takes two vector subspaces F and E and outputs a vector of $E \setminus F$, e.g. by testing all the vectors of a basis of E.
– Algorithm POINT_IS_NOT_IN takes a vector x and a vector space F and outputs TRUE if and only if $x \notin F$. This can be done by Gaussian elimination.

6.2 Construction of the Schedule Linear Parts

Preprocessing. For each statement S we complete $\{\mathbf{F}_S^1, ..., \mathbf{F}_S^{d_S}, \mathbf{C}_S^1, ..., \mathbf{C}_S^\delta\}$ in a set of n linearly independent vectors using some vectors $\mathbf{L}_S^1, ..., \mathbf{L}_S^{n-\delta-d_S}$. We build the matrix $\mathbf{M}_{S,0}$ below, where $\mathbf{F}_S$, resp. $\mathbf{L}_S$, resp. $\mathbf{C}_S$, is the matrix whose i-th row vector is the vector $\mathbf{F}_S^i$, resp. $\mathbf{L}_S^i$, resp. $\mathbf{C}_S^i$. For each graph $G_u(S,i)$ we build (e.g. by rational linear programming [5]) a solution $\mathbf{P}(S,i)$ of the system:

$$\begin{cases} e = (x_e, y_e) \in (G_u(S,i))' \Rightarrow \mathbf{P}(S,i)\boldsymbol{w}(e) + \rho_{y_e} - \rho_{x_e} \geq 0 \\ e = (x_e, y_e) \notin (G_u(S,i))' \Rightarrow \mathbf{P}(S,i)\boldsymbol{w}(e) + \rho_{y_e} - \rho_{x_e} \geq 1 \end{cases} \quad (3)$$

$$\mathbf{M}_{S,0} = \begin{pmatrix} \mathbf{F}_S \\ \mathbf{L}_S \\ \mathbf{C}_S \end{pmatrix}$$

We want all statements included in $G_u(S,i)$ to have the same i-th linear part, and this linear part to be a point of $\mathsf{P}(S,i)$. For that, we add to the i-th row of each matrix $\mathbf{M}_{T,i}$ the same adequate number of times the vector $\mathbf{P}(S,i)$.

Algorithm to Build Linear Parts in $\mathsf{P}(S,i)$ from Linear Parts in $\mathsf{VS}(S,i)$

For $i = 1$ **to** $\max_{S \in G_u} d_S$ **do**

- **For each** subgraph $G_u(S,i)$ **do**
  1. Find an integer ν s.t. $(\mathbf{F}_S^i + \nu\, \mathbf{P}(S,i))$ belongs to $\mathsf{P}(S,i)$, i.e. s.t. there exist some constants ρ_S satisfying for each edge $e = (x_e, y_e) \in G_u(S,i)$:

     $$\begin{cases} e \in G_u(S,i)' \text{ or } x_e \text{ is virtual} & \Rightarrow (\mathbf{F}_S^i + \nu \mathbf{P}(S,i))\boldsymbol{w}(e) + \rho_{y_e} - \rho_{x_e} \geq 0 \\ e \notin G_u(S,i)' \text{ and } x_e \text{ is actual} & \Rightarrow (\mathbf{F}_S^i + \nu \mathbf{P}(S,i))\boldsymbol{w}(e) + \rho_{y_e} - \rho_{x_e} \geq 1 \end{cases}$$

  2. **For each** T in $G_u(S,i)$, let $\mathbf{M}'_{T,i-1}$ be equal to $\mathbf{M}_{T,i-1}$ plus the vector $\mathbf{P}(S,i)$ on the i-th row. Let $\gamma_T = \det(\mathbf{M}_{T,i-1})$ and $\gamma'_T = \det(\mathbf{M}'_{T,i-1})$.
  3. Compute the set: $\Gamma = \left\{ \frac{-\gamma_T}{\gamma'_T - \gamma_T} \,\middle|\, T \in G_u(S,i), \gamma_T \neq \gamma'_T \right\}$
  4. Let λ be an integer s.t. $\lambda \geq \nu$ and $\lambda \notin \Gamma$. **For each** statement T of $G_u(S,i)$, let $\mathbf{M}_{T,i}$ be equal to $\mathbf{M}_{T,i-1}$ plus λ times the vector $\mathbf{P}(S,i)$ on the i-th row. Condition $\lambda \geq \nu$ ensures that the i-th row of $\mathbf{M}_{T,i}$ belongs to $\mathsf{P}(T,i)$, while condition $\lambda \notin \Gamma$ ensures that $\mathbf{M}_{T,i}$ is non singular.

For each statement S, the first d_S rows of the matrix $\mathbf{P}_{S,d_S}$ define the d_S linear parts $\mathbf{E}_S^1, ..., \mathbf{E}_S^{d_S}$ of its schedule.

Note: the missing proofs and explanations can be found in [4].

6.3 Computation of the Constants

We have the linear parts of our schedule but not yet the constants. To build them, we process each graph $G_u(S,i)$ as follows (see [5, Section 7.1.2]):

1. Weight any edge $e = (x_e, y_e)$ of $G_u(S,i)$ with a new weight $\boldsymbol{w}'(e) = X\boldsymbol{w}(e) - l(e)$, where $l(e) = 1$ if x_e is actual and if $e \notin (G_u(S,i))'$, and $l(e) = 0$ otherwise.
2. Add a node S_0 and a zero-weight edge from S_0 to each node of $G_u(S,i)$.
3. Use a shortest path algorithm and let the constant ρ_S^i be the opposite of the weight of the shortest path from S_0 to S.

6.4 Algorithm Complexity

Algorithm Build_Vectors has a complexity of $O(s^2n^4(n+s^2))$. The building of the linear parts has a complexity of $O(sn^4 + Z)$, where Z is the complexity of Darte-Vivien (see [5] for details). For the constant computations see [5].

7 Conclusion

We have presented an algorithm that produces, for a perfect loop nest, an affine scheduling compatible with a given affine mapping of its computations. When the representation of the dependences is a polyhedral approximation of distance vectors, our algorithm succeeds whenever such an affine schedule exists. The cases of success are defined by a necessary and sufficient condition which can easily be checked. In this paper, in order to simplify things (!), we limited ourselves by using an approximation of dependences by polyhedra. But with a few tricks [4], our method can actually be extended to Feautrier's algorithm [8]. This algorithm works on an exact representation of dependences. Exact dependence analysis is feasible for static control programs with affine array access functions, which is the only type of programs most mapping algorithms work with.

References

[1] J. R. Allen and K. Kennedy. Automatic translation of Fortran programs to vector form. *ACM TOPLAS*, 9(4):491–542, Oct. 1987.

[2] J. M. Anderson and M. S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices*, 28(6):112–125, June 1993.

[3] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill. Solving alignment using elementary linear algebra. In K. Pingali, U.Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and compilers for parallel computing - 7th international workshop*, volume LNCS 892, pages 46–60. Springer Verlag, 1994.

[4] A. Darte, C. Diderich, M. Gengler, and F. Vivien. Scheduling the computations of a loop nest with respect to a given mapping. Technical Report 00-04, ICPS, University of Strasbourg, France, 2000.

[5] A. Darte and F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *Int. J. Parallel Programming*, 25(6), 1997.

[6] C. G. Diderich and M. Gengler. The alignment problem in a linear algebra framework. In *Proceedings of the Hawaï International Conference on System Sciences (HICSS-30), Software Technology Track*, pages 586–595, Wailea, HI, Jan. 1997. IEEE Computer Society Press.

[7] M. Dion and Y. Robert. Mapping affine loop nests: New results. In B. Hertzberger and G. Serazzi, editors, *High-Performance Computing and Networking, International Conference and Exhibition*, volume LCNS 919, pages 184–189. Springer-Verlag, 1995.

[8] P. Feautrier. Some efficient solutions to the affine scheduling problem, part II: multi-dimensional time. *Int. J. Parallel Programming*, 21(6):389–420, 1992.

[9] P. Feautrier. Towards automatic distribution. *Parallel Processing Letters*, 4(3):233–244, 1994.

[10] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* ACM Press, 1997.

[11] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE TPDS*, 2(4):472–482, Oct. 1991.

[12] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE TPDS*, 2(4):452–471, Oct. 1991.