Affine-by-Statement Scheduling of Uniform Loop Nests over Parametric Domains

Alain Darte and Yves Robert*

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

May 7, 1992

1 Introduction

In this report we deal with affine-by-statement scheduling, a high-level technique for the parallelization of loop nests with uniform dependences. Affine-by-statement scheduling can be viewed as a natural extension of Lamport's hyperplane method [4], which has been proposed by many authors, including Rao [10], Quinton [7] and Rajopadhye [9]. For the sake of self-completeness, we briefly review affine-by-statement scheduling in the next section.

Given a loop nest, a method for determining the best affine-by-statement scheduling has been proposed by Darte and Robert [3]. The underlying idea is to use the duality theorem of linear programming and to solve an optimization problem, e.g. via the simplex method, to find the optimal parameters.

Although constructive, the method in [3] suffers from several drawbacks. Roughly speaking, solving the optimization problem is very costly, since it depends upon the total number of dependences between statements. Even more, for parametric computation domains (think of a matrix-matrix product of size N, where N is the parameter), the solution is only possible at

*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.

execution-time (when N is known) whereas we would like to compute it at compile-time.

It is the objective of this paper to remedy these drawbacks. We delay the precise formulation of our results up to section 2.6, because we need to introduce several hypotheses and definitions beforehand.

2 Affine-by-statement scheduling

2.1 Definition and example

We review affine-by-statement scheduling of loop nests in this section. 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. If there is a dependence between iteration $S_i(I)$ and iteration $S_j(J)$, the dependence vector between these iterations 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

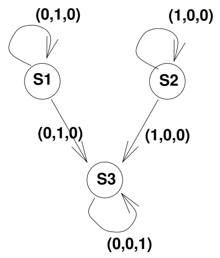

Figure 1: The dependence graph of matrix-matrix product

edges corresponding to the dependence vectors. The computation domain is represented by the integer points contained within a convex $\{x, Ax \leq b\}$.

Consider the very simple example of the uniformized matrix-matrix product whose dependence graph is shown figure 1. 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
            $S_1 : a(i,j,k) = a(i,j-1,k)$
            $S_2 : b(i,j,k) = b(i-1,j,k)$
            $S_3 : c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$
        enddo
    enddo
enddo

In the following, we deal with uniform loop nests, which we express in terms of their structural information, namely:

- the computation domain represented linearly by the set of points $\{x, Ax \leq b\}$.
- the dependence vectors represented by the $d_{i,j}$.

Scheduling a uniform loop nest for parallel execution amounts 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 $p \in E(time)$ do in parallel (*do all*)

```
        P(p)
    enddo
enddo
```

The external loop corresponds to an iteration time and $E(time)$ is the set of all the points computed at step $time$. Lamport [4] proposes to search for an admissible scheduling 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 then 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.

Rather than considering the k statements of program P as a whole block, we can try to schedule them separately. A natural generalization of Lamport's hyperplane method 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 $\lfloor {}^t\pi_i.p + c_i \rfloor$.

To illustrate the difference between Lamport's method and affine-by-statement scheduling, we go back to the matrix-matrix product example. If we look for a unique scheduling vector π for the 3 statements, we must ensure that $\pi D \geq 1$, where $D = \begin{pmatrix} d_1 & d_2 & d_3 \end{pmatrix}$, with $d_1 = (1,0,0)$, $d_2 = (0,1,0)$ and $d_3 = (0,0,1)$. We obtain $\pi = (1,1,1)$ and a parallel execution time equal to $3n$.

Using affine-by-statement scheduling, we can have the following execution scheme:

- 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 $2n$.

2.2 General optimization problem

How can we express dependence constraints for affine-by-statement scheduling ? Assume that node p of statement S_i is executed at time $\lfloor X_i.p + c_i \rfloor$:

$$
\begin{cases}
\forall i, 1 \le i \le k \\
X_i = X_i' A \\
X_i = -X_i'' A \\
X_i' \ge 0 \\
X_i'' \ge 0 \\
X_i D_i \ge 1 \\
\\
\forall i, 1 \le i \le k \\
\forall j \in Pred(i) \\
X_j - X_i = X_{j,i} A \\
X_{j,i} \ge 0 \\
-X_{j,i} b + X_j d_{j,i} + c_i - c_j \ge 1 \\
\\
\forall i, 1 \le i \le k \\
\gamma_1 - X_i' b - c_i \ge 0 \\
\gamma_2 - X_i'' b + c_j \ge 0 \\
\\
\min(\gamma_1 + \gamma_2)
\end{cases}
$$

Figure 2: General problem

- if statement S_i has self-dependences represented by the matrix D_i, then we must have $X_i D_i \ge 1$.
- if there is a dependence from statement S_i to statement S_j, with vector $d_{i,j}$, then we must have for each point p such that $Ap \le b$:

$$X_i p + c_i - X_j(p - d_{j,i}) - c_j \ge 1$$

Determining the best affine-by-statement scheduling is not easy. Darte and Robert [3] show that it can be done by solving the linear problem given figure 2. We call this problem the **general problem**. Here is a proof of this result.

Theorem 1 *The best affine-by-statement schedules are the optimal solutions of the general problem (figure 2).*

Proof : We first show that any solution of this problem is a valid affine-by-statement schedule, because it preserves the dependences. The condition

$X_i D_i \geq 1$ implies that the self-dependences are preserved, whereas the conditions:

$$\left\{\begin{array}{l} X_j - X_i = X_{j,i} A \\ X_{j,i} \geq 0 \\ -X_{j,i} b + X_j d_{j,i} + c_i - c_j \geq 1 \end{array}\right.$$

imply that for any p in the domain (satisfying $Ap \leq b$), we have:

$$\begin{array}{rcl} X_i p + c_i - X_j(p - d_{j,i}) - c_j & = & -(X_j - X_i)p + X_j d_{j,i} + c_i - c_j \\ & = & -X_{j,i} Ap + X_j d_{j,i} + c_i - c_j \\ & \geq & -X_{j,i} b + X_j d_{j,i} + c_i - c_j \\ & \geq & 1 \end{array}$$

thus preserving the dependences between statements. In the same way, $\gamma_1 + \gamma_2$ is an upper bound of the total time of the schedule. Indeed, for all p in the domain, for all i,

$$\begin{array}{rcl} X_i p + c_i & = & X_i' Ap + c_i \\ & \leq & X_i' b + c_i \\ & \leq & \gamma_1 \\ X_i q + c_i & = & -X_i' Ap + c_i \\ & \geq & -X_i'' b + c_i \\ & \geq & -\gamma_2 \end{array}$$

Now, consider an optimal affine-by-statement schedule, given by X_i and c_i. We have to show that it is a solution of the general problem and thus find suitable X_i', X_i'', $X_{j.i}$, γ_1 and γ_2. Consider the following linear problem for a fixed i:

$$\max_{(p, Ap \leq b)} X_i p$$

This maximum exists because the time given by X_i is bounded and is equal to the minimum of the dual problem:

$$\left\{\begin{array}{l} X_i' A = X_i \\ X_i' \geq 0 \\ \min(X_i' b) \end{array}\right.$$

This gives X_i'. By a similar argument, we obtain X_i'' by considering

$$\max_{(p, Ap \leq b)} -X_i p$$

and $X_{j,i}$ by considering

$$\max_{(p, Ap \leq b)} (X_j - X_i)p$$

Furthermore, we have for all p such that $Ap \leq b$:

$$X_i p + c_i - X_j(p - d_{j,i}) - c_j \geq 1$$

which can be successively rewritten:

$$\begin{array}{l}
\min_{(p,Ap \leq b)}(X_i - X_j)p + X_j d_{j,i} + c_i - c_j \geq 1 \\
-\max_{(p,Ap \leq b)}(X_j - X_i)p + X_j d_{j,i} + c_i - c_j \geq 1 \\
-X_{j,i} b + X_j d_{j,i} + c_i - c_j \geq 1
\end{array}$$

Finally, we choose $\gamma_1 = \max_i(X_i' b + c_i)$ and $\gamma_2 = -\min_i(-X_i'' b + c_i)$. γ_1 is the time of the last computation and $-\gamma_2$ the time of the first one. Thus, $\gamma_1 + \gamma_2$ represents the total time of the schedule. □

2.3 Parametric computation domains

Many loop nests define computation domains whose size depends upon the size of the problem or the size of the data. For instance, in matrix computations, the problems are often parametrized by the size of the matrices that are used. Our goal is to be able to schedule loop nests at compile-time, even if some information about the computation domain is still unknown. We want the compilation time to depend maybe upon the dimension of the problem, but not upon its size. This is to guarantee that, for large problem instances, the compilation time will be small compared to the execution time. For this reason, we will deal with parametric computation domains. We recall here the notions of unit computation domain, of fatness of a domain and of parametric domains which were introduced in [2].

The computation domain of a loop nest can be represented by a domain $\{Ax \leq b\}$. A is the constraint matrix and intuitively gives the shape of the domain whereas b gives the bounds on the domain, i.e. gives an idea of its size. We use the following notation: if $P = \{Ax \leq b, x \in \mathcal{R}^n\}$, then tP denotes the polyhedron $\{Ax \leq tb, x \in \mathcal{R}^n\}$. We make some realistic assumptions on the computation domains to avoid degenerate cases which never occur in practice.

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. We note this condition* $0 + \Box \in P$.

The first condition is just a technical condition which consists in shifting the domain so that it contains the origin 0, without loss of generality. Note that it implies that $b \geq 0$. Condition (2) means that the domain contains at least one pattern of the dependence system, which is a reasonable condition for practical cases. Actually, in this paper, we will not need this condition. Condition (3) implies that the domain is not too thin nor too skewed. In practice, computation domains are often squares, cubes, triangles, or they can have more complicated shapes but they do not look like needles.

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

Now, considering a fat domain $\{Ax \leq b\}$ and a uniform loop nest 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. If $\mathcal{D} = \{Ax \leq b\}$ has fatness 1 (this is what we call a **unit domain**), then N coincides with the fatness.

2.4 Scheduling parametric domains

If we want to deal with domains like $\{Ax \leq Nb\}$, we have to replace, in the linear problem of figure 2, b by Nb, and to find a way to solve such a parametric linear problem. However, if we look at the results obtained for different large N, it is obvious that such solutions are not suitable for rewriting the loop.

Indeed, consider the uniform loop nest whose domain is the square $Sq(N) = \{p = (i,j), 0 \leq i, j \leq N\}$ and whose dependence graph is given in figure 3. Note that $Sq(N) = \{Ax \leq Nb\}$, with

$$A = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \text{ and } b = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

We obtain the following optimal solutions:

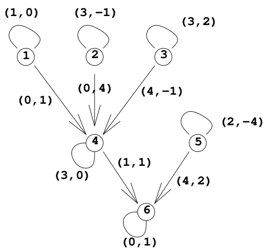

Figure 3: The dependence graph of the tree-like example over $Sq(N)$

With $N = 100$

$$\begin{array}{ll}
X_1 = (1, \frac{100}{101}) & X_2 = (\frac{42}{101}, \frac{25}{101}) \\
X_3 = (\frac{301}{1111}, \frac{104}{1111}) & X_4 = (1, \frac{100}{101}) \\
X_5 = (\frac{10001}{10100}, \frac{4951}{20200}) & X_6 = (\frac{10001}{10100}, 1) \\
c_1 = c_2 = c_3 = c_6 = \frac{-1}{101} & c_4 = 0, c_5 = \frac{6951}{2020} \\
time = \frac{20101}{101} \simeq 199 &
\end{array}$$

With $N = 10000$

$$\begin{array}{ll}
X_1 = (1, \frac{10000}{10001}) & X_2 = (\frac{4167}{10001}, \frac{2500}{10001}) \\
X_3 = (\frac{30001}{110011}, \frac{10004}{110011}) & X_4 = (1, \frac{10000}{10001}) \\
X_5 = (0, \frac{-1}{4}) & X_6 = (0, 1) \\
c_1 = c_2 = c_3 = \frac{-1}{10001} & c_4 = 0, c_5 = \frac{199969997}{20002} \\
c_6 = \frac{100000000}{10001} & time = \frac{200010001}{10001} \simeq 19999
\end{array}$$

With $N = 1000000$

$$\begin{array}{ll}
X_1 = (1, \frac{1000000}{1000001}) & X_2 = (\frac{416667}{1000001}, \frac{250000}{1000001}) \\
X_3 = (\frac{3000001}{11000011}, \frac{1000004}{11000011}) & X_4 = (1, \frac{1000000}{1000001}) \\
X_5 = (0, \frac{-1}{4}) & X_6 = (0, 1) \\
c_1 = c_2 = c_3 = \frac{-1}{1000001} & c_4 = 0, c_5 = \frac{1999996999997}{2000002} \\
c_6 = \frac{1000000000000}{1000001} & time = \frac{2000001000001}{1000001} \simeq 1999999
\end{array}$$

It would be more significant to find the solution $X_1 = X_4 = (1, 1), X_2 = (\frac{5}{12}, \frac{1}{4}), X_3 = (\frac{3}{11}, \frac{1}{11}), X_5 = (0, \frac{-1}{4}), X_6 = (0, 1)$ and $c_1 = c_2 = c_3 = c_4 = 0, c_5 = N - \frac{3}{2}, c_6 = N$ whose execution time is $2N$. Actually, we will show that there always exists an "asymptotic solution" independent of N, and with a time nearly optimal when N grows to infinity.

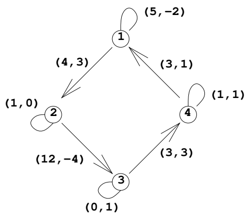

Figure 4: The dependence graph of the cyclic example over $Sq(N)$

2.5 Scheduling cycles

The number of inequalities in the general optimization problem is proportional to the number of edges in the dependence graph, which renders its solution very lenghty.

However, the following example gives intuitive evidence that it is possible to reduce the complexity: consider the uniform loop nest whose domain is the square $Sq(N)$ and and whose dependence graph is given in figure 4. We obtain the following optimal solution:

With $N = 10000$

$$\begin{array}{ll} X_1 = (1, \frac{1249000700172509}{1250000299939991}) & X_2 = (1, \frac{4993003999489901}{5000001199759964}) \\ X_3 = (\frac{1249124600010006}{1250000299939991}, 1) & X_4 = (\frac{1249749337449997}{1250000299939991}, \frac{1249375562500000}{1250000299939991}) \\ c_1 = c_2 = c_3 = c_4 = 0 & time = \frac{24991248999499970000}{1250000299939991} \simeq 19993 \end{array}$$

Again, it would be better to be able to find the solution $X_1 = X_2 = X_3 = X_4 = (1,1)$ whose execution time is $2N$. We will show that for each cycle in the dependence graph, the scheduling vectors for all the statements within the cycle can be chosen identical in the asymptotic solution mentioned before.

2.6 New results

After setting all our material, we can state the main achievements of this paper:

- the introduction of a **limit problem** that enables to determine quasi-optimal affine-by-statement schedules for parametric computation domains at compile-time.

$$
\left\{
\begin{array}{l}
\forall i, 1 \leq i \leq k \\
X_i = X_i^{'} A \\
X_i = -X_i^{''} A \\
X_i^{'} \geq 0 \\
X_i^{''} \geq 0 \\
X_i D_i \geq 1 \\
\\
\forall i, 1 \leq i \leq k \\
\forall j \in Pred(i) \\
X_j - X_i = X_{j,i} A \\
X_{j,i} \geq 0 \\
-X_{j,i} b + c_i' - c_j' \geq 0 \\
X_j d_{j,i} + c_i^* - c_j^* \geq 1 \\
\\
\forall i, 1 \leq i \leq k \\
\gamma_1 - X_i' b - c_i' \geq 0 \\
\gamma_2 - X_i'' b + c_j' \geq 0 \\
\\
\min(\gamma_1 + \gamma_2)
\end{array}
\right.
$$

Figure 5: Limit problem

- the proof that optimal scheduling vectors for statements lying in the same strongly connected component of the dependence graph are identical in the limit problem, thereby greatly reducing the complexity of the optimization problem to be solved.

3 Limit problem

In this section, we deal with parametric computation domains $\{Ax \leq Nb\}$, and we show the existence of a solution which is independent of N although asymptotically optimal.

Consider the linear problem given figure 5. We call this problem the **limit problem**. Any of its solutions leads to a valid schedule on the domains $\{Ax \leq Nb\}$ if we consider that point p of statement S_i is executed at time $\lfloor X_i p + N c_i' + c_i^* \rfloor$. This is because any solution $(X_i, c_i', c_i^*), 1 \leq i \leq k$ is also a solution of the problem of figure 2 with $c_i = Nc'_i + c^*_i$.

We point out that the limit problem is non empty. A possible solution is to choose all the X_i to be equal to Lamport's vector (see [4]) and to take all constants c'_i and c^*_i equal to 0. The total execution time of a solution of the limit problem is

$$N(\gamma_1 + \gamma_2) + \max_i(c^*_i) - \min_i(c^*_i)$$

thus is equivalent to $N(\gamma_1 + \gamma_2)$ when N is large. Note $K = \max(c^*_i) - \min(c^*_i)$, $T_\infty = \gamma_1 + \gamma_2$ for the optimal solution of the limit problem and T_N the time of the optimal solution for the general problem with parameter N. Thus, we have:

$$\forall N \qquad T_N \leq N T_\infty + K \tag{1}$$

Theorem 2 *T_N/N has a limit and*

$$\lim_{N\to\infty} \frac{T_N}{N} = T_\infty$$

Proof : First, we prove that, when N is large, the different steps of the simplex method do not depend on N. Indeed, with the notations of Schrijver (see [11, p.133-134]), the tests are done on (equation (16))

$$(u'_k, u''_k) := -(c, 0) + c_B B_k^{-1}[A\ I]$$

and the solution is constructed by choosing (equation (22))

$$\lambda_0 = \min\{(B_k^{-1}b)_t/(B_k^{-1}g)_t | t = 1, \ldots, m; (B_k^{-1}g)_t > 0\}$$

These expressions are rational fractions in N. They keep a constant sign when N is large, actually as soon as N is larger than their largest pole. Thus, the behavior of the simplex algorithm does not depend on N when N is large and a solution can be parametrized as a rational fraction in N.

The total execution time is greater than the time of any equation considered independently. If X is the scheduling vector of a given equation, for all N sufficiently large, $T_N \geq \max X(p - q)$ with p and q in $N\mathcal{D}$. Now, we use the fact that $0 + \Box \in \mathcal{D}$. Ne_i is in $N\mathcal{D}$. Thus, $T_N \geq NX^i$, where X^i is the i-th component of X. This implies that X^i is a rational fraction in N of degree at most 0, because T_N is $O(N)$ from equation 1.

Since the c_i are defined up to a translation, we can consider than the execution of the algorithm starts at time 0. Then, for all $p \in \mathcal{D}$ and for all i, $X_i p + c_i$ is $O(N)$. This proves that the c_i are rational fraction in N of degree at most 1 (take for example $p = 0$).

Now, we know that X_i has a limit $\tilde{X}_i$ when N grows to ∞, (the same for X_i' and X_i'') and that c_i/N has also a limit, say $\tilde{c}_i'$ (same thing for γ_1/N and γ_2/N). Inequalities and equalities holds for the limit solution.

$$-X_{j,i}Nb + X_j d_{j,i} + c_i - c_j \geq 1$$

becomes, by dividing by N and making N tend towards infinity,

$$-\tilde{X}_{j,i}b + \tilde{c}_i' - \tilde{c}_j' \geq 0$$

and

$$\begin{cases} \gamma_1 - X_i' Nb - c_i' \geq 0 \\ \gamma_2 - X_i'' Nb + c_j' \geq 0 \end{cases}$$

becomes

$$\begin{cases} \tilde{\gamma}_1 - \tilde{X}_i' b - \tilde{c}_i' \geq 0 \\ \tilde{\gamma}_2 - \tilde{X}_i'' b + \tilde{c}_j' \geq 0 \end{cases}$$

The $\tilde{X}_i$, $\tilde{X}_i'$, $\tilde{X}_i''$, $\tilde{c}_i'$, $\tilde{\gamma}$ will form a solution of the limit problem provided that we are able to find $\tilde{c}_i^*$ such that

$$\tilde{X}_j d_{j,i} + \tilde{c}_i^* - \tilde{c}_j^* \geq 1$$

The existence of such $\tilde{c}_i^*$ comes from the retiming theorem of Leiserson and Saxe [5]. We have seen in section 2.2 that for all p in $\mathcal{D}$,

$$X_i p + c_i - X_j(p - d_{j,i}) - c_j \geq 1$$

On a cycle of the dependence graph, we have by adding the corresponding inequalities:

$$\sum_{cycle} X_{i,succ(i)} d_{i,succ(i)} \geq \text{cycle length}$$

This remains true for the limit solution $\tilde{X}$. Then, the retiming theorem proves the existence of the constants $\tilde{c}_i^*$, and provides a constructive way of determining them. Thus, we have found a solution of the limit problem with time $\lim_{N\to\infty} T_N/N$. This gives a lower bound for $\lim_{N\to\infty} T_N/N$.

$$\lim_{N\to\infty} T_N/N \quad \geq \quad T_\infty \tag{2}$$

Thus, with equations 1 and 2, $\lim_{N\to\infty} \frac{T_N}{N} = T_\infty$ □

Solving the limit problem can be done without knowing N, thus at compile-time, and we have shown that it provides a solution, whose execution time is equivalent to the time of the best affine-by-statement schedule, when N is large. Furthermore, we will show that the limit problem has some nice properties which allow us to solve it much more easily than the original general problem.

4 Scheduling cycles of the dependence graph

In this section, we show that for the limit problem, all scheduling vectors on a cycle of the dependence graph are identical.

Theorem 3 *In a cycle, all X_i are equal, and all c'_i are equal.*

Proof : Consider a cycle of length l in the dependence graph describing the loop nest and note $Y_i = X_{i,succ(i)}$. Then, by adding the l inequalities:

$$-X_{i,succ(i)}b + c'_{succ(i)} - c'_i \geq 0$$

we obtain

$$-(\sum_{i=1}^{l} Y_i)b \geq 0$$

Now, $Y_i \geq 0$ and $b \geq 0$, thus $(\sum_{i=1}^{l} Y_i)b \geq 0$. This implies $(\sum_{i=1}^{l} Y_i)b = 0$ and then $Y_i b = 0$. So, for all i, $c'_{succ(i)} - c'_i \geq 0$. This shows that all c'_i are equal. Permute the basis vectors so that b can be rewritten in the simpler form:

$$b = \begin{pmatrix} b_1 \\ 0 \end{pmatrix} \quad Y_i = (0, Y_i^2) \quad A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$$

The domain $\mathcal{D}$ is now defined by $\{A_1 x \leq b_1 \text{ and } A_2 x \leq 0\}$. For $0 + \square \in \mathcal{D}$, for all i, $A_2 e_i \leq 0$: the column vectors of A_2 are non positive. Thus, for all i, $Y_i^2 A_2 \leq 0$. Because $Y_i A = Y_i^2 A_2$ we have:

$$\begin{aligned} (\sum_{i=1}^{l} Y_i^2)A_2 &= (\sum_{i=1}^{l} Y_i)A \\ &= \sum_{i=1}^{l}(X_i - X_{succ(i)}) \\ &= 0 \end{aligned}$$

and thus, for all i, $Y_i^2 A_2 = 0$ which gives $X_i = X_{succ(i)}$. $\square$

This shows that the limit problem can be simplified by replacing for each strongly connected component of the dependence graph constitued by r statements, the search for r schedule vectors and constants c' by the search for only one vector and one constant. Only the constants c^* remain possibly distinct. This is what happens in the example of section 2.5: we may obtain the solution

$$X_1 = X_2 = X_3 = X_4 = 1, c_1^* = 13, c_2^* = 7, c_3^* = 0, c_4^* = 16$$

5 Practical implementation

As a consequence of the results of the previous sections, we can state the following simplified version of the limit problem, given in figure 6, in order to schedule uniform loop nests. In the dependence graph, the strongly connected components are identified and denoted by $SCC(i)$ for $1 \leq i \leq r$. This induces a new (reduced) graph with strongly connected components as nodes and precedence constraints between components.

A further simplified version of the limit problem would come from a step-by-step reduction based upon a topological sort of the reduced dependence graph: first compute the optimal schedule for nodes without predecessors, and then iterate for those nodes whose predecessors have all been solved. Unfortunately, this approach does not yield optimal solutions, as the very simple following example demonstrates.

Consider the uniform loop nest whose domain is the square $Sq(N) = \{p = (i,j), 0 \leq i,j \leq N\}$ and whose dependence graph is given in figure 7. If we solve the limit problem, we obtain the solution

$$X_1 = X_2 = (0,1), c_1' = c_2' = c_1^* = c_2^* = 0$$

hence an execution time of N steps.

However, if we solve step-by-step, there are several optimal solutions for scheduling node 1: in particular, we can choose either $X_1 = (1,0)$ or $X_1 = (0,1)$ (and $c_1' = c_1^* = 0$). But the choice has dramatic effects on the scheduling of node 2: if we take $X_1 = (0,1)$ we get the previous solution but with $X_1 = (1,0)$ we obtain $X_2 = (0,1), c_1' = c_1^* = 1$ and a total execution time of $2N$ steps.

$$
\left\{
\begin{array}{l}
\text{One scheduling vector by strongly connected component:} \\
\forall i, 1 \leq i \leq r \\
X_i = X_i' A \\
X_i = -X_i'' A \\
X_i' \geq 0 \\
X_i'' \geq 0 \\
\forall j \in SCC(i) \\
X_i D_j \geq 1 \\
\\
\text{Dependence constraints between components:} \\
\forall i, 1 \leq i \leq r \\
\forall j \in Pred(i) \\
X_j - X_i = X_{j,i} A \\
X_{j,i} \geq 0 \\
-X_{j,i} b + c_i' - c_j' \geq 0 \\
\\
\text{Dependence constraints between statements:} \\
\forall k, l, k \neq l \text{ such that } d_{k,l} \text{ exists} \\
k \in SCC(j), l \in SCC(i) \\
X_j d_{k,l} + c_l^* - c_k^* \geq 1 \\
\\
\text{Constraints on } \gamma_1 \text{ and } \gamma_2 \\
\forall i, 1 \leq i \leq r \\
\gamma_1 - X_i' b - c_i' \geq 0 \\
\gamma_2 - X_i'' b + c_j' \geq 0 \\
\\
\min(\gamma_1 + \gamma_2)
\end{array}
\right.
$$

Figure 6: Final problem with r strongly connected components

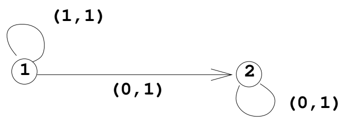

Figure 7: The dependence graph of the 2-node example over $Sq(N)$

6 Conclusion

In this report we have dealt with affine-by-statement scheduling for uniform loop nests over parametric domains. The main results are the following:

- the introduction of a **limit problem** that enables to determine quasi-optimal affine-by-statement schedules at compile-time.
- the proof that optimal scheduling vectors for statements lying in the same strongly connected component of the dependence graph are identical in the limit problem, thereby greatly reducing the complexity of the optimization problem to be solved.

Appendix

A1. Shifted domains

In the proofs of the theorems, we needed the conditions $0 \in \mathcal{D}$ and $0+\square \in \mathcal{D}$. Actually, we only need that a shifted copy of $\mathcal{D}$ is fat: there exists p_0 such that $\mathcal{D} - p_0$ is fat. With such assumptions, the proof of the first theorem remains correct with the two following modifications:

- $p_0 + \square \in \mathcal{D}$ implies that $p = Np_0$ and $q = Np_0 + Ne_i$ are in $N\mathcal{D}$. So each component of X_i is $O(1)$, because the total execution time is $O(N)$.
- After possibly translating the constant c_i so that the algorithm starts after 0, we get that c_i is $O(N)$ because $X_i N p_0 + c_i$ is $O(N)$.

Now consider the two linear problems: problem 1, the limit problem with b and problem 2, the limit problem with $b' = b - Ap_0$. Note that now $\{Ax \leq b'\}$ is fat and so theorem 2 is true. Remark that from any solution of problem 1, we get a solution of problem 2, just by changing c'_i into $c'_i + X_i p_0$. Thus, X_i is constant on any cycle of the reduced graph. It follows also that even in problem 1, the c_i are constant on a cycle. Thus, theorem 2 is also true for shifted copies of fat domains.

This property is important in practice because we do not need to compute p_0 before solving the limit problem.

A2. On the constants c_i^*

Our aim is to be able to rewrite the nested loop into a parallel loop as regular as possible. For this, we would like to use the index-shift method of [6] on the c_i^*. In a strongly connected component scheduled with X and c', a point p of statement i is scheduled at time $\lfloor Xp + Nc' + c_i^* \rfloor$. We choose a vector u_i such that $Xu_i = c_i^*$ and we translate the indices of the loop so that the new point p' is computed at time $\lfloor X(p' - u_i) + Nc' + c_i^* \rfloor$, i.e $\lfloor Xp' + Nc' \rfloor$. This simplification permits to rewrite in a single loop all statements of the same strongly connected component. However, to do this transformation, we need u to be an integral vector. Write $X = (1/\lambda)Y$ where Y is an integer vector with co-prime components. Take u such that $Y.u = 1$ and let $u_i = (\lambda c_i^*)u$. u_i is solution if λc_i^* is an integer. This may be false in the solution found when solving the final problem. However, such c_i^* can be found by the following process:

1. For each strongly connected component, select a particular node (the "host") and apply the retiming theorem of Leiserson and Saxe by computing r_i as the shortest path from the host to node i, considering that the weight of an edge (j, i) is $Xd_{j,i} - 1$. Then all r_i satisfy by construction that λr_i is an integer and $Xd_{j,i} + r_i - r_j \geq 1$ (cf for example [8][p.258]).

2. Sort the reduced graph (strongly connected components as nodes) by a topological sort. Note that this graph has no cycle which makes the following step work.

3. Construct the c_i^* component by component as follows: if the current component has no predecessor, let $c_i^* = r_i$. Otherwise choose the smallest integer k such that, by letting $c_i^* = r_i + \lambda k$, all the constraints between the current component and its predecessors are satisfied.

The time of the new solution differs by a small constant from the previous one.

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] Alain Darte and Yves Robert. Scheduling uniform loop nests. Technical Report 92-10, LIP, Ecole Normale Supérieure de Lyon, France, February 1992. submitted to IEEE Trans. PDS.

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

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

[6] 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.

[7] 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.

[8] Patrice Quinton and Yves Robert. *Systolic Algorithms and Architectures*. Prentice Hall and Masson, 1989.

[9] 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.

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

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