McGill University
School of Computer Science

Software Pipelining for Nested Loops

ACAPS Technical Memo 53
(This report also appears as EPPP Project Technical Report 1)
May 5, 1993

Guang R. Gao and Qi Ning

School of Computer Science
McGill University
Montreal, Quebec
Canada H3A 2A7

Vincent Van Dongen

Centre de Recherche Informatique de Montreal
Montreal, Quebec H3T 1P1

Advanced Compilers, Architectures and
Parallel Systems Group

Contents

List of Figures

Abstract

In this paper, we present a novel framework of software pipelining for nested loops. Under this framework, a periodic scheduling function, called *r-periodic schedule*, is associated with each operation of the loop body in the entire iteration space. We present a simple problem formulation as well as efficient solutions which gives provable asymptotically time-optimal schedule for nested loops under our program model. It also provides some useful insights on the interplay between unimodular loop transformation and fine-grain scheduling under a unified framework.

The main results are:

- We show that, for nested loops under our program model, software pipelining problem can be formulated into a very simple linear programming problem based on *r-periodic schedules*.
- We prove that the r-periodic scheduling is *asymptotically time-optimal*, that is, it is asymptotically as good as the best schedule — the *free* schedule where statements can be *freely* scheduled as early as possible subject only to data dependence constraints.
- When the nesting level n is small, we present an efficient polynomial time solution method. This surprisingly efficient solution is based on a vectorized extension of the solution of the *minimum cost-to-time ratio cycle* (MCTRC) problem [29].
- It is often desirable to use a small fixed periodicity r in the schedule. We present an efficient solution to handle this situation.

A discussion of related and future work is included.

1 Introduction

Loop scheduling such as *software pipelining* plays an important role in the design of optimizing compilers for high-performance computer architectures. Software pipelining has been successfully applied to high-performance pipelined processor architectures, as well as super-scalar and VLIW architectures [2, 3, 4, 20, 21, 27, 32, 36, 37, 40]. However, most proposed software pipelining methods have been focused on one-dimensional loops (or the innermost loop of a nested loop). These methods were primarily targeted to uniprocessor architectures with limited instruction level parallelism. Little attention has been paid to the scheduling problem of nested loops.

However, we have witnessed the fast evolution of parallel computer architectures in the past few years. Microprocessors with multimillion transistors are in their horizon, supporting unprecedented opportunity of ample instruction level parallelism. Multiprocessor machines based on the connection of many nodes will continue to progress rapidly. As research into parallel architectures continues to make progress, more powerful and scalable parallel machines will become readily available with competitive cost/performance level.

Parallelism of innermost loops alone may no longer be enough to keep these machines usefully busy. In other word, we need to look at ways of extending software pipelining to nested loops such that parallelism among operations from different iterations of the entire multi-dimensional iteration space can be exploited.

There have been substantial progress in various loop transformation techniques for parallelizing compilers [6, 45]. Unimodular transformation has been proposed as a method to exploit parallelism in nested loops [7, 44]. It provides a framework to combine other loop transformation techniques such as *loop interchange*, *loop skewing* and *loop reversal*. However, the existing work in unimodular transformation has not been targeted on time-optimal loop scheduling. In fact, statement timing has not even been considered explicitly as a parameter in the problem formulation. In this paper, we demonstrate that the optimal scheduling of loops cannot be obtained through unimodular loop transformation without properly considering the timing information. Furthermore, we show that isolating the unimodular transformation from the scheduling phase may also fail to obtain an optimal schedule. Therefore, we present a framework which captures both loop transformation and instruction scheduling in an unified mathematical framework.

The goal of this paper is to develop a framework for multi-statement multi-dimensional loop scheduling. In this framework, we propose to use a periodic scheduling function for each operation of the loop body in the entire iteration space. In one-dimensional case [43], the schedule takes the form

$$t(i, v) = \lfloor \frac{T \cdot i + A[v]}{r} \rfloor,$$

where the parameters T, r and $A[v]$ are integers, and $t(i, v)$ is the starting time of the i^{th} occurrence of the v^{th} statement of the loop body. (Throughout this paper, $\lfloor \frac{a}{b} \rfloor$ denotes the floor of $\frac{a}{b}$, i.e. the largest integer less or equal to $\frac{a}{b}$, while $\lceil \frac{a}{b} \rceil$ is the ceil of $\frac{a}{b}$, i.e. the smallest integer larger or equal to $\frac{a}{b}$.) In this form, the schedule is periodic with periodicity r, period T and offset time $A[v]$. This formulation does not have the restriction that the schedule of each individual iteration must be the same. In fact, the schedule of two consecutive iterations may not be the same, but will repeat every r iterations; hence called *r-periodic*.

In this paper, we demonstrate that such r-periodic scheduling can be extended for nested loops, by generalizing T and i into n-dimensional vectors where n is the dimension of the loop. The main results are, for loops defined on rectangular regions and with non-negative coefficient dependence vectors:

- We show that, using the form of r-periodic schedule, the time-optimal multi-dimensional software pipelining problem can be formulated into a linear programming problem based on r-periodic schedules.

- We prove that the r-periodic scheduling is *asymptotically time-optimal*, that is, it is asymptotically as good as the best schedule — the *free* schedule where statements can be scheduled as early as possible subject only to data dependence constraints.

- When the nesting level n is small, we present an efficient polynomial time solution method. This surprisingly simple solution can be considered as a vector extension of the solution of the *minimum cost-to-time ratio cycle* (MCTRC) problem [29]. Our extended algorithm performs a multi-dimensional search while the solution of the MCTRC problem performs a binary search. The key intuition leading to this efficient solution is the observation that the time-optimal computation rate of a multi-dimensional loop is dominated by its *critical dependence cycles*.

  We also illustrate how to transform the loop according to the computed schedule.

- It may often be desirable to use a small fixed periodicity r in the schedule. We show that the optimal scheduling problem for a fixed r is NP-complete. However, in real life applications, the dimension of loops to be scheduled is small, hence an efficient solution is possible — and we present such a method based on our search algorithm mentioned earlier.

Our method provides a uniform framework that combines *unimodular loop transformation*, loop unrolling and statement level scheduling for multiple statements nested loops. Generalization of our techniques and a comparison with other related work are also discussed.

1.1 Synopsis

This paper is organized as follows. We present examples in Section 2 to motivate our approach. We introduce our program model in Section 3. In Section 4, we show that the periodic scheduling problem can be formulated as a linear programming problem. We prove that the periodic scheduling is *asymptotically time-optimal*, i.e., its difference from a free scheduling is only a constant independent of the loop bounds. In Section 5 we present our new search algorithm to solve the optimal periodic scheduling problem with low dimensions very efficiently. We also illustrate how to perform loop transformation based on the computed schedule. Section 6 focuses on the problem of periodic scheduling with a fixed periodicity. We show in this section that the fixed r-periodic scheduling problem is NP-complete. We also extend our algorithm to solve the fixed r-periodic scheduling problem for loops with low dimensions. In Section 7, we discuss related and future work.

2 Motivating Examples

In this section, we use three examples to illustrate the notion of r-periodic scheduling and how it can handle software pipelining for nested loops. All these loops contain multiple statements in their body.

The first example is taken from [9]. It illustrates how useful it might be to consider both loop parallelization and statement scheduling in a unified framework. In example

2.2, the optimal scheduling of the loop cannot be obtained simply through unimodular loop transformation as described in [8, 28, 44]. Instead, it requires a unimodular transformation followed by instruction scheduling[1] to obtain the optimal schedule. We demonstrate, in intuitive terms, that the r-periodic scheduling captures both loop transformation and instruction scheduling in an unified mathematical framework.

In Example 2.3, we present a loop for which even performing the unimodular transformation plus instruction scheduling together cannot reach the optimal schedule. Intuitively, the time-optimal schedule of the loop is dominated by the critical dependence cycle [23], which is not treated by the traditional instruction scheduling technique. We show that such cycle constraints are captured elegantly by the periodicity in the framework of the proposed r-periodic scheduling.

Example 2.1 Consider the following loop taken from [9].

$$\begin{array}{ll} & \text{for } i = 0 \text{ to } N_1 - 1 \\ & \quad \text{for } j = 0 \text{ to } N_2 - 1 \\ & \quad\quad \text{begin} \\ S_1: & \quad\quad\quad a(i,j) = b(i,j-1); \\ S_2: & \quad\quad\quad b(i,j) = a(i-1,j) + a(i-1,j-1); \\ & \quad\quad \text{end} \end{array} \tag{1}$$

In [9], they first transform the loop into a parallel form and then perform instruction scheduling. The final code is unfortunately not optimal because both loop parallelization and instruction scheduling are not considered together.

In this paper, we present a systematic technique to compute the optimal schedule in one step. The corresponding schedule is always periodic. For loop (2.1), we can prove that the optimal periodic schedule is:

$$t((i,j),v) = 2i + A(v),$$

with $A(1) = 1$ and $A(2) = 0$. Here, $t((i,j),v)$ represents the starting time of statement S_v at iteration (i,j). After rewriting, the loop becomes:

$$\begin{array}{l} \text{for } i = 0 \text{ to } N_1 - 1 \\ \quad \text{forall } j = 0 \text{ to } N_2 - 1 \\ \quad\quad \text{begin} \\ \quad\quad\quad b(i,j) = a(i-1,j) + a(i-1,j-1); \\ \quad\quad\quad a(i,j) = b(i,j-1); \\ \quad\quad \text{end} \end{array}$$

In this case, a simple reordering of S_1 and S_2 transforms the nested loop into one whose inner loop is parallel. In addition to be optimal, this loop is much simpler to

[1]For our purpose, the terms statement and instruction are interchangeable here.

implement than the one derived in [9].

□

Example 2.2 Let λ be a positive integer and consider the following loop.

$$
\begin{array}{ll}
 & \text{for } i = 0 \text{ to } N \\
 & \quad \text{for } j = 0 \text{ to } N \\
 & \quad\quad \text{begin} \\
S_1 : & \quad\quad a(i,j) = f_1(b(i-1, j-\lambda), c(i-1, j+\lambda), ...); \\
S_2 : & \quad\quad b(i,j) = f_2(a(i, j-1), ...); \\
S_3 : & \quad\quad c(i,j) = f_3(a(i, j-1), ...); \\
 & \quad\quad \text{end}
\end{array}
\tag{2}
$$

□

We assume that each of the statements S_1, S_2 and S_3 takes one unit of time to get executed. So, the total sequential execution time to execute the loop is:

$$3(N+1)^2$$

The dependence graph (DG) of this loop, when $\lambda = 5$, is shown in Figure 1.

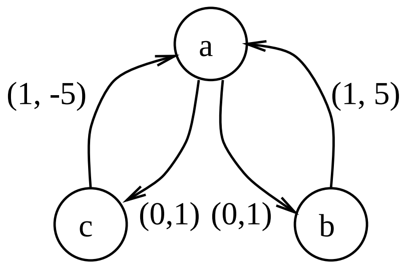


Figure 1: Dependence graph for the loop (2) when $\lambda = 5$.

Let us first consider the use of a unimodular transformation to parallelize the loop. Proposed compile time unimodular transformation for loops [8, 44] treats the loop body (3 statements in our example) as a single unit, and the transformation that minimizes the total execution time is shown below:

$$\begin{pmatrix} t \\ p \end{pmatrix} = \begin{pmatrix} 1+\lambda & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix}. \tag{3}$$

With such a transformation, the loop is rewritten as:

$$
\begin{array}{l}
\text{for } t = 0 \text{ to } (2+\lambda)N \\
\quad \text{forall } p = max(0, \lceil \frac{t-N}{1+\lambda} \rceil) \text{ to } min(N, \lfloor \frac{t}{\lambda+1} \rfloor) \\
\quad\quad \text{begin} \\
\quad\quad\quad x = t - p(\lambda+1); \\
\quad\quad\quad a(p,x) = f_1(b(p-1, x-\lambda), c(p-1, x+\lambda), ...); \\
\quad\quad\quad b(p,x) = f_2(a(p, x-1), ...); \\
\quad\quad\quad c(p,x) = f_3(a(p, x-1), ...); \\
\quad\quad \text{end}
\end{array}
\tag{4}
$$

Note that we have introduced an additional variable x that simplifies the expressions of the indices. Let us simply neglect the additional computation time of this statement. Then, each loop body takes three units of time to get executed. Therefore, the total execution time for this loop is:

$$3((2+\lambda)N+1).$$

Compared with the serial execution, we get a factor of improvement of:

$$\frac{3(N+1)^2}{3((2+\lambda)N+1)}$$

which is of the order of N/λ. Obviously, the larger λ is the less the speedup.

One can parallelize this loop by further applying statement level scheduling technique to the transformed loop body, and the result is the following:

$$
\begin{array}{l}
\text{for } i = 0 \text{ to } N \\
\quad \text{forall } j = 0 \text{ to } N \\
\quad\quad \text{begin} \\
\quad\quad\quad a(i,j) = f_1(b(i-1, j-\lambda), c(i-1, j+\lambda), ...); \\
\quad\quad\quad b(i,j) = f_2(a(i, j-1), ...) || \\
\quad\quad\quad c(i,j) = f_3(a(i, j-1), ...); \\
\quad\quad \text{end}
\end{array}
$$

This loop body takes two units of time to get executed because S_2 and S_3 are performed in parallel (as denoted by "||"). The total execution time of the loop is therefore:

$$2(N+1),$$

which is much improved from the one obtained previously with the unimodular transformation.

For $N = 2$, the statements are scheduled as shown in Figure 2. The arcs in the graph represent the data dependences between the computations, when $\lambda = 2$. Clearly, such a schedule is valid in the sense that all the dependence constraints are satisfied. Furthermore, it can be shown that it minimizes the total execution time.

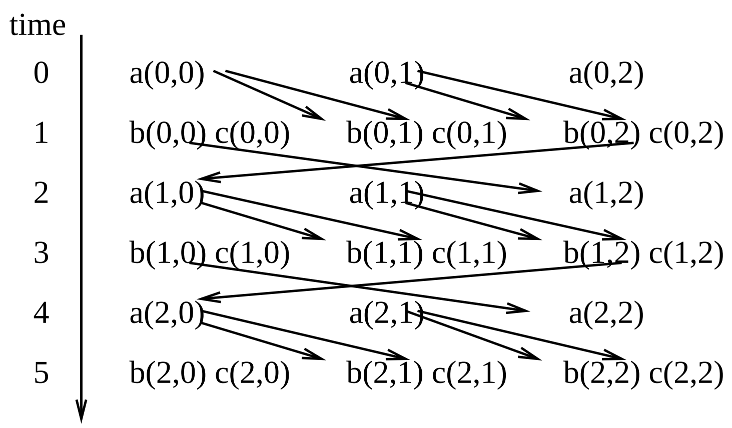


Figure 2: The execution of the optimal parallel loop.

The factor of improvement compared with the serial loop (2) is:

$$\frac{3(N+1)^2}{2(N+1)} = \frac{3(N+1)}{2}.$$

The factor of improvement compared with the parallel loop obtained with the unimodular transformation is:

$$\frac{3((2+\lambda)N+1)}{2(N+1)},$$

which is of the order of λ. The larger λ is, the larger the improvement is. The additional parallelism obtained from this loop cannot, therefore, be bounded *a priori* compared with the one obtained with a unimodular transformation alone.

In this paper, we present a method which formulate the problem of deriving the ultimate time-optimal schedule in a unified mathematical framework. We propose a periodic schedule of the form (for a two-dimensional loop)

$$t((i,j),v) = \lfloor \frac{T_i \cdot i + T_j \cdot j + A(v)}{r} \rfloor, \tag{5}$$

that minimizes the total execution time. Here, $t((i,j),v)$ represents the starting time of statement S_v at iteration (i,j); T_i and T_j are integers, $A(v)$ is a positive integer and r is a positive integer. Let us illustrate the meaning of these parameters using Example 2.

In terms of (5), a unimodular transformation makes use of a schedule of the form:

$$t((i,j),v) = T_i \cdot i + T_j \cdot j,$$

where T_i and T_j are relatively prime integers. That is, the effect of unimodular transformation is captured by the vector — the affine scheduling vector. This form, clearly a restriction of (5), ignores both r and the parameters $A(v)$'s. Intuitively, the $A(v)$'s specifies the relative offset of the statements of the body in a periodic schedule. That is, they capture exactly what the instruction scheduling phase is trying to optimize in our example. Therefore, $A(v)'s$ are essential in order to fully exploit the fine-grain parallelism of a multiple-statement loop. For our example, the best schedule is given by:

$$t((i,j),v) = 2 \cdot i + A(v),$$

with $A(1) = 0$ and $A(2) = A(3) = 1$. So, this means that:

- The loop can be parallelized along the index j. Indeed, because $(T_i, T_j) = 2(1,0)$, we can apply a unimodular transformation equal to the identity matrix, and the inner loop can be parallelized.
- The statements S_2 and S_3 can be scheduled in parallel, after S_1. (This is what the values of $A(v)$'s say.)
- A new iteration of the serial loop can start every two time units. (This is because the greatest common divisor of (T_i, T_j) is 2.

Clearly, the schedule in the form of (5) expresses the unimodular transformation and the scheduling of the loop statements together in a unified mathematical form.

Example 2.3

Even combined unimodular transformations and instruction scheduling may not be sufficient to exploit all the parallelism in loop. Consider the following loop.

$$\begin{array}{ll}
 & \text{for } i = 0 \text{ to } N \text{ do} \\
 & \quad \text{for } j = 0 \text{ to } N \text{ do} \\
 & \quad\quad \text{begin} \\
S_1: & \quad\quad\quad a(i,j) = a(i-1,j) + c(i-1,j); \\
S_2: & \quad\quad\quad b(i,j) = a(i,j) + b(i,j-1); \\
S_3: & \quad\quad\quad c(i,j) = b(i-1,j) + c(i-1,j); \\
 & \quad\quad \text{end}
\end{array} \tag{6}$$

It can be shown that, in terms of r-periodic scheduling, the following is the optimal schedule:

$$t((i,j),v) = \lfloor \frac{3i + 2j + A(v)}{2} \rfloor,$$

with $A(1) = 0$, $A(2) = 2$ and $A(3) = 1$.

Note that the parameter r can no longer be ignored in this example. Figure 3 shows the dependence graph of the loop. We can observe that the critical cycle has a total dependence distance of (2,0). Intuitively, this suggests the fact that the optimal schedule should contains two instances of each statement on the first coordinate (i). Figure 4

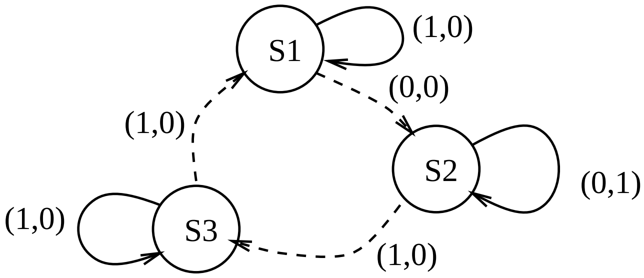


Figure 3: The dependence graph for the loop (6) and its critical cycle shown in dash.

shows the optimal scheduling of the instructions at time 6 and time 7, and the data dependences accordingly.

□

t	t0								
6	12	a(4,0)	c(3,1)	b(2,2)	a(2,3)	c(1,4)	b(0,5)	a(0,6)	
	13	c(4,0)	b(3,1)	a(3,2)	c(2,3)	b(1,4)	a(1,5)	c(0,6)	
7	14	b(4,0)	a(4,1)	c(3,2)	b(2,3)	a(2,4)	c(1,5)	b(0,6)	a(0,7)
	15	a(5,0)	c(4,1)	b(3,2)	a(3,3)	c(2,4)	b(1,5)	a(1,6)	c(0,7)

Figure 4: Optimal scheduling at time 6 and time 7 for the loop (6).

In the next two sections, we formalize the intuition developed through these examples, and present efficient solution to compute the optimal schedule.

3 Program Model

In this section, we introduce the basic program model for loops treated in this paper. In Section 7, we discuss the limitations of the model, its extensions, as well as other related work.

Our program model consists of a class of *perfectly nested* loops. To simplify the model, we require that the loops are *normalized* and have the form:

$$
\begin{array}{l}
\text{for } i_1 = 0 \text{ to } N_1 \\
\quad \text{for } i_2 = 0 \text{ to } N_2 \\
\quad \vdots \\
\qquad \text{for } i_n = 0 \text{ to } N_n \\
\qquad \quad \text{begin} \\
\qquad \quad \; S_1; \\
\qquad \quad \; S_2; \\
\qquad \quad \; \vdots \\
\qquad \quad \; S_m; \\
\qquad \quad \text{end}
\end{array}
$$

The loop has the following properties:

- It has a unit step size.
- Each of the m statements in the loop body, S_v $(1 \leq v \leq m)$, is an assigment statement defining an array element. Let $\mathbf{i}$ be the index vector $(i_1, i_2, ..., i_n)$, all the statements in the body are of the form:

$$a(\mathbf{i}) = f(b(\mathbf{i} - \delta), ...) \tag{7}$$

  where

  - a and b are two arrays of $N_1 \times N_2 \times ... \times N_n$ entries computed by two statements in the loop.
  - δ is a constant integer vector with non-negative components.
  - Without loss of generality, f is some *elementary function* (e.g. unary or binary operations) whose arguments are either external inputs or are of the form $b(\mathbf{i} - \delta)$.

The dependence information of a loop is represented by its *dependence graph* (DG). A statement S_v is represented as a node, and the dependence between statements is represented by a directed edge (arc) between them. We will assume that the DG is strongly connected. Each node u is labeled by a non-negative number $d(u, v)$ which represents the time it takes to execute S_v. When all statements are unit-delays, $d(u, v) = 1$. In our loops, an arc is labeled with a corresponding *dependence distance* $\delta(u, v)$. When δ is a non-zero vector, we call the corresponding dependence a *loop-carried dependence*. The total number of arcs of the dependence graph is then in $\mathcal{O}(m)$, where m is the number of statements in the loop body.

4 Periodic Scheduling and its Optimality

4.1 Periodic schedules

Given a nested loop whose index $(i_1, i_2, \cdots, i_n)$ is of dimension n, and whose body has m instructions, we use the notation $t((i_1, i_2, \cdots, i_n), v)$ to indicate the starting time clock to schedule statement S_v in the iteration indexed by $(i_1, i_2, \cdots, i_n)$. We will be interested in finding a schedule of the following form:

$$t((i_1, i_2, \cdots, i_n), v) = \lfloor \frac{T_1 \cdot i_1 + T_2 \cdot i_2 + \cdots + T_n \cdot i_n + A(v)}{r} \rfloor,$$

whose parameters $T_1, T_2, \cdots T_n$ are integers, $A(1), A(2), \cdots, A(m)$ are non-negative integers, and r is a positive integer. In vector form, the schedule is written as:

$$t(\mathbf{i}, v) = \lfloor \frac{\mathbf{T} \cdot \mathbf{i} + A(v)}{r} \rfloor, \tag{8}$$

where $\mathbf{i} = (i_1, i_2, \cdots i_n)$ and $\mathbf{T} = (T_1, T_2, \cdots, T_n)$.

By definition, we will call a schedule of the form (8) *r-periodic*, and call $\mathbf{T}$ and r its *period* and periodicity respectively. See Section 2 for examples of periodic schedules. Below, we give the r-periodic schedule for the loop (3) in Section 2.

Example 2.2 (cont'd) For loop (4), the associated r-periodic schedule (8) is:

$$t((i, j), v) = 3 \cdot ((1 + \lambda) \cdot i + j) + v.$$

Note that here r=1.

□

For one-dimensional loops, the scheduling vector degenerates to a scalar, so does the iteration vector. The readers are referred to Section 7 for a discussion of related work.

4.2 A Linear Programming Formulation

In this section, we formulate the optimal r-periodic scheduling problem into a linear programming problem.

Among the starting times of all statements, let $max_{\mathbf{i},v}(t(\mathbf{i}, v))$ and $min_{\mathbf{i},v}(t(\mathbf{i}, v))$ denote the latest one and earliest one, respectively. Let ΔT be $max_{\mathbf{i}}(\mathbf{T} \cdot \mathbf{i})/r - min_{\mathbf{i}}(\mathbf{T} \cdot \mathbf{i})/r$ and let ΔA be $max_v(A(v))/r - min_v(A(v))/r$. Let Δt be total execution time of a periodic schedule, then we have:

$$\Delta t = \Delta T + \Delta A - \epsilon,$$

where $\epsilon \in [0,1]$. It is easy to see that ΔT depends on the size of the domain of $\mathbf{i}$ while ΔA does not. Therefore, asymptotically for large domains, it is important to minimize ΔT. Let us define the **fastest periodic schedule** as the one that minimizes ΔT. The following theorem formulates the problem of finding the fastest periodic schedule into a linear programming problem.

Theorem 4.1 *Given a loop as defined in Section 3, the fastest periodic schedule can be found by minimizing*

$$T_1 \cdot N_1/r + T_2 \cdot N_2/r + ... + T_n \cdot N_n/r \tag{9}$$

under the constraints:

$$\mathbf{T} \cdot \delta(u,v) + A(v) - A(u) \geq r \cdot d(u,v),\ \forall \delta(u,v) \tag{10}$$

$$T_j \geq 0,\ \forall j \in [1,n] \tag{11}$$

Proof A schedule of the form (8) must obey all the data dependences between pairs of instructions, that is to say, if (u,v) is a dependence arc in the DG then we have the constraint:

$$t(\mathbf{i},v) - t(\mathbf{i} - \delta(u,v),u) \geq d(u,v), \tag{12}$$

where $d(u,v)$ is the delay on dependence arc (u,v), and $\delta(u,v)$ its dependence distance vector.

A constraint (12) can be expanded as:

$$\lfloor \frac{\mathbf{T} \cdot \mathbf{i} + A(v)}{r} \rfloor - \lfloor (\frac{\mathbf{T} \cdot (\mathbf{i} - \delta(u,v)) + A(u)}{r} \rfloor) \geq d(u,v). \tag{13}$$

We now prove that (13) is true if and only if the following is true in which the floor functions are taken out from (13):

$$\frac{\mathbf{T} \cdot \mathbf{i} + A(v)}{r} - (\frac{\mathbf{T} \cdot (\mathbf{i} - \delta(u,v)) + A(u)}{r}) \geq d(u,v) \tag{14}$$

Clearly, for any integers a, b, c, d and x, if $\frac{a}{b} - \frac{c}{d} \geq x$, then $\lfloor \frac{a}{b} \rfloor - \lfloor \frac{c}{d} \rfloor \geq x$. Therefore, (14) implies (13).

In [41], it is proven that, for any integers d and c, the average value of the function

$$x \in \mathbf{Z} \rightarrow f(x) = \lfloor \frac{x}{d} \rfloor - \lfloor \frac{x-c}{d} \rfloor$$

is c/d. Thus, the average value of the function $f(\mathbf{i})$ defined by (13) is $(\mathbf{T} \cdot \delta(u,v) + A(v) - A(u))/r$. Constraint (14) can be simplified into the following form:

$$(\mathbf{T} \cdot \delta(u,v) + A(v) - A(u))/r \geq d(u,v)$$

(which is exactly (10)). If (10) is not true, then (13) won't be true for some values of $\mathbf{i}$. This finishes the proof that (13) is true if and only if (14) (and therefore (10)) is true.

Next let us justify the nonnegative constraints on $\mathbf{T}$. Suppose that there exists an optimal $\mathbf{T}$ in which some of its components are negative. Without loss of generality, we can assume that $T_1, \cdots, T_k$ are nonnegative, and $T_{k+1}, \cdots, T_n$ are negative (where $1 \leq k < n$) . Now let us consider the total time span of the schedule function t in terms of this given $\mathbf{T}$. Since the index domain is rectangular, the largest possible value of $t(\mathbf{i}, v)$ can be obtained by making the first k indices taken their largest values and the rest $n-k$ indices taken their smallest value (zero), that is:

$$T_1N_1 + \cdots + T_kN_k$$

Similarly, the smallest value of t can be obtained by letting the first k indices taken their smallest values (zero) and letting the rest $n-k$ indices taken their largest values, i.e.:

$$T_{k+1}N_{k+1} + \cdots + T_nN_n$$

which is a negative number. So the total time span is:

$$T_1N_1 + \cdots + T_kN_k - (T_{k+1}N_{k+1} + \cdots + T_nN_n).$$

Let $\mathbf{T}'$ be a vector defined as follows:

$$T_i' = T_i \text{ if } T_i \geq 0, \text{ and } T_i' = 0 \text{ if } T_i < 0.$$

Thus all the components of the new $\mathbf{T}'$ are nonnegative. It is easy to verify that the newly defined $\mathbf{T}'$ is still a feasible solution of the constraints (10). However the total time span of the new $\mathbf{T}'$ is shorter than that of $\mathbf{T}$, because the smallest value of t in terms of $\mathbf{T}'$ is now 0 and the largest value of t in terms of $\mathbf{T}'$ is the old largest value of t in terms of $\mathbf{T}$. This contradicts the assumption that $\mathbf{T}$ is optimal and proves that all the components of the vector $\mathbf{T}$ should be nonnegative. With this in mind, it is easy to see that the total time span is from 0 to

$$T_1N_1 + \cdots + T_nN_n$$

which is the objective function in (9).

∎

4.3 Asymptotic Optimality

This section gives us another key motivation for using periodic schedules. It proves that periodic schedules not only have a simple linear programming formulation, but also do not compromise optimality when compared with other best schedules.

In the following, let $\mathbf{N} = (N_1, N_2, ..., N_n)$, let ΔT_f be the total time it takes to execute a loop when using the free schedule (i.e. the statements are executed as soon as possible) and let ΔT_p be the total time it takes to execute a loop when using the fastest periodic schedule. We will say that a periodic schedule is *asymptotically time-optimal* if and only if $(\Delta T_p - \Delta T_f)$ is a constant independent on $\mathbf{N}$, i.e. this periodic schedule is as fast as the free schedule for a sufficiently large value of $\mathbf{N}$. The main result of this section is presented in the following theorem.

Theorem 4.2 *Given a loop as defined in Section 3, its fastest periodic schedule is asymptotically time-optimal.*

Before we present the proof of the main theorem we first sketch our proof plan, and state the role of its supporting lemmas.

- The basic intuition and observation behind the proof is that there exists a critical dependence path along which the affine schedule cannot be improved over the free schedule more than by a constant. Such a critical dependence path is captured by *dependence cycles* in the dependence graph. Lemma 1 formulates the constraints on dependence cycles in an equivalent linear programming formulation.

- Furthermore, the direction of the optimization, which determines the objective function of the optimal solution, can also be related with the dependence cycles, and in its most general form, may be specified as a linear combination of n fundamental cycles. Lemma 2 has established this fact.

Let us first state Lemma 1 which points out that the $\mathbf{T}$ in the linear programming problem (9), (10) and (11) only depends on the cycles in the dependence graph DG.

Lemma 1 *A* $\mathbf{T}$ *will admit a feasible solution for the* $A(u)$*'s in the LP problem (9), (10) and (11) if and only if for each cycle* C *in the dependence graph DG,* $\mathbf{T}$ *satisfies the following constraint:*

$$T_1 \sum_{(u,v)\in C} \delta_1(u,v) + \cdots + T_n \sum_{(u,v)\in C} \delta_n(u,v) \geq r \cdot \sum_{(u,v)\in C} d(u,v). \tag{15}$$

The proof of the above lemma can be found, for instance, in [19].

Lemma 2 *If the optimal* $\mathbf{T}$ *is determined by two cycles* c_1 *and* c_2*, that is,*

$$\begin{array}{c} \mathbf{T} \cdot \sum_{(u,v)\in c_1} \delta(u,v) = r \cdot \sum_{(u,v)\in c_1} d(u,v) \\ \mathbf{T} \cdot \sum_{(u,v)\in c_2} \delta(u,v) = r \cdot \sum_{(u,v)\in c_2} d(u,v) \end{array} \tag{16}$$

then the direction of optimization $\mathbf{N}$ *is in the cone generated by* $\mathbf{C}_1$ *and* $\mathbf{C}_2$*, where* $\mathbf{C}_1 = \sum_{(u,v)\in c_1} \delta(u,v)$ *and* $\mathbf{C}_2 = C_2 = \sum_{(u,v)\in c_2} \delta(u,v)$.

Proof Let

$$d_1 = \sum_{(u,v)\in c_1} d(u,v) \quad \text{and} \quad d_2 = \sum_{(u,v)\in c_2} d(u,v).$$

Consider the following dual of the LP problem (20), (21), (22):

$$\max \sum_{\text{for all cycle } c_i \text{ in DG}} \alpha(c_i) \tag{17}$$

subject to

$$\sum_{\text{for all cycle } c_i \in DG} \alpha(c_i)\mathbf{C_i} \le \mathbf{N} \tag{18}$$

$$\alpha(c_i) \ge 0, \forall c_i \in DG \tag{19}$$

¿From linear programming theory, we can find an optimal solution to this dual problem so that there are at most two dual variables α being positive and all the other dual variables being zero. Let this two positive dual variables be α_1 and α_2 and they correspond to cycles c_1 and c_2. We argue here these two variables also make the constraint (18) as an equality. This actually follows from the complementary slackness because both primal variables T_1 and T_2 are assumed to be strictly positive by the lemma. This completes the proof of the lemma.

■

And now we are ready to give the proof of Theorem 4.2.

Proof of Theorem 4.2.

Without loss of generality, let us assume that $n = 2$. (The proof can be easily generalized to the case when $n \neq 2$.) Let us consider the affine schedule $t_a(\mathbf{i}, v) = \frac{\mathbf{T}\cdot\mathbf{i}+A[v]}{r}$. It is easy to show that $t(\mathbf{i}, v)$ is asymptotically time-optimal if and only if $t_a(\mathbf{i}, v)$ is asymptotically time-optimal.

By Lemma 1, $\mathbf{T}$ can be found by solving the following LP problem:

$$\min N_1T_1 + N_2T_2 \tag{20}$$

subject to

$$T_1 \sum_{(u,v)\in C} \delta_1(u,v) + T_2 \sum_{(u,v)\in C} \delta_2(u,v) \ge r \sum_{u\in C} d(u,v). \tag{21}$$

$$T_i \ge 0, i = 1, 2 \tag{22}$$

The possible feasible solutions for the above LP program are the vertices of the polyhedron defined by constraints (21) and within the first quadrant enclosed by the two axes as specified by (22). Therefore, there are two cases of the possible optimal solutions for the above LP program.

Case I: The first possibility is that the solution is a vertex of the polyhedron not on the two axes: i.e. both components of $\mathbf{T}$ are positive. Then it is a solution of a system determined by two cycles c_1 and c_2 as in (16). Recall that $\mathbf{C}_1 = \sum_{(u,v)\in c_1} \delta(u,v)$ (resp. $\mathbf{C}_2 = \sum_{(u,v)\in c_2} \delta(u,v)$) and $d_1 = \sum_{(u,v)\in c_1} d(u,v)$ (resp. $d_2 = \sum_{(u,v)\in c_2} d(u,v)$). By Lemma 2, $\mathbf{N}$ is in the cone generated by $\mathbf{C}_1$ and $\mathbf{C}_2$.

Without any restriction, we can assume that $\mathbf{C}_1$ passes through statement S_1. In general, $\mathbf{C}_1$ and $\mathbf{C}_2$ may not have common nodes of the DG as shown in Figure 5.

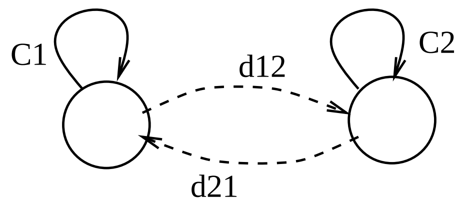


Figure 5: Example of a DG with two simple cycles $\mathbf{C}_1$ and $\mathbf{C}_2$ passing through different nodes.

Because the DG is strongly connected, there always exists a path between them. Let $\delta_{1,2}$ (resp. $\delta_{2,1}$) be the sum of the dependence vectors along the path going from S_1 (resp. S_2) to $S_k \in \mathbf{C}_2$ (resp. some $S_k \in \mathbf{C}_2$).

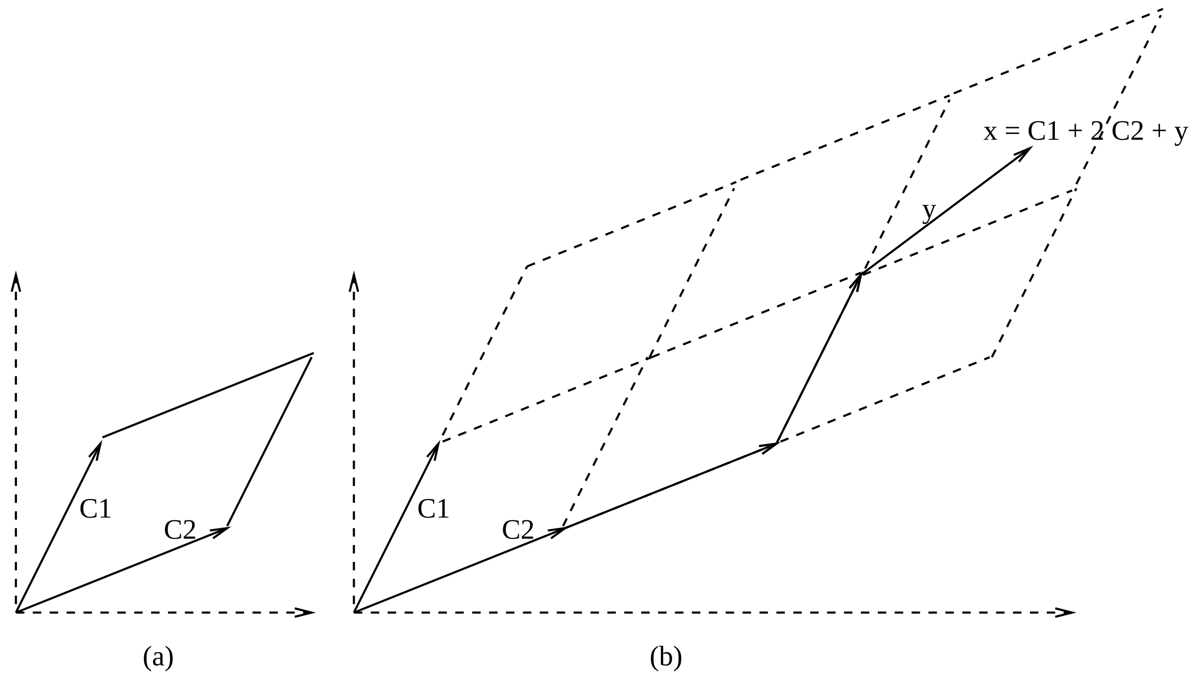


Figure 6: (a) The parallelogram of edges $\mathbf{C}_1$ and $\mathbf{C}_2$ and (b) a vector $\mathbf{x}$ expressed as a non-negative integer combination of $\mathbf{C}_1$ and $\mathbf{C}_2$ plus a small vector.

Because of Theorem 4.1, $\Delta T_p - t_a(\mathbf{N}, 1)$ is a small constant independent on $\mathbf{N}$. We would like to find an iteration vector $\mathbf{N}'$ in the iteration space such that

- $\mathbf{N} - \mathbf{N}'$ is a small vector whose size is independent of $\mathbf{N}$,
- and there is a (true) dependence path in the completely unfolded dependence graph between statement S_1 at iteration $(0, 0, ..., 0)$ and S_1 at iteration $\mathbf{N}'$ along which the affine schedule cannot be improved over the free schedule more than by a constant independent of $\mathbf{N}$.

Indeed, if this is the case, $\Delta T_p - t_a(\mathbf{N}', 1)$ is a constant and so is $t_a(\mathbf{N}', 1) - \Delta T_f$. This would prove the asymptotic time optimality of the periodic schedule.

We now show that such a point $\mathbf{N}'$ exists. Let us construct a parallelogram of edges $\mathbf{C}_1$ and $\mathbf{C}_2$ as shown in Figure 6(a), and denote all the integer vectors inside this parallelogram as $\mathbf{y}_1$, $\mathbf{y}_2$, ..., $\mathbf{y}_g$. Let us consider any point $\mathbf{x}$ inside the cone formed by $\mathbf{C}_1$ and $\mathbf{C}_2$ in the unfolded dependence graph (see Figure 6). Point $\mathbf{x}$ can be expressed as:

$$\mathbf{x} = \alpha_1(\mathbf{x})\mathbf{C}_1 + \alpha_2(\mathbf{x})\mathbf{C}_2 + \mathbf{y}_{f(\mathbf{x})}, \tag{23}$$

where $\alpha_1(\mathbf{x})$ and $\alpha_2(\mathbf{x})$ are non-negative integers, $f(\mathbf{x}) \in [1, g]$. See Figure 6

Now let

$$\mathbf{N}'' = (\mathbf{N} - \delta_{1,2} - \delta_{2,1}), \tag{24}$$

and let $\mathbf{N}$ be sufficiently large so that $\mathbf{N}''$ is also inside the cone formed by $\mathbf{C}_1$ and $\mathbf{C}_2$. Thus, this vector can be expressed as

$$\mathbf{N}'' = \alpha_1(\mathbf{N}'')\mathbf{C}_1 + \alpha_2(\mathbf{N}'')\mathbf{C}_2 + \mathbf{y}_{f(\mathbf{N}'')}.$$

In other words,

$$\mathbf{N} = \alpha_1(\mathbf{N}'')\mathbf{C}_1 + \delta_{1,2} + \alpha_2(\mathbf{N}'')\mathbf{C}_2 + \delta_{2,1} + \mathbf{y}_{f(\mathbf{N}'')}. \tag{25}$$

Let $\mathbf{N}' = \mathbf{N} - \mathbf{y}_{f(\mathbf{N}'')}$. In other words,

$$\mathbf{N}' = \alpha_1(\mathbf{N}'')\mathbf{C}_1 + \delta_{1,2} + \alpha_2(\mathbf{N}'')\mathbf{C}_2 + \delta_{2,1}. \tag{26}$$

For $\mathbf{N}$ sufficiently large, $\mathbf{N}'$ is inside the domain. Because of (26), there is a dependence path in the unfolded dependence graph from the origin to $\mathbf{N}'$ which is composed of (possibly multiple of) $\mathbf{C}_1$, $\mathbf{C}_2$, $\delta_{1,2}$ and $\delta_{2,1}$. Due to (16), we have:

$$\Delta T_f \geq \alpha_1(\mathbf{N}'')d_1 + \alpha_2(\mathbf{N}'')d_2,$$

That is even the free schedule cannot speedup the schedule of N'. Furthermore,

$$t_a(\mathbf{N}', 1) = \alpha_1(\mathbf{N}'')d_1 + \alpha_2(\mathbf{N}'')d_2 + \mathbf{T}(\delta_{1,2} + \delta_{2,1})/r.$$

Thus, $t_a(\mathbf{N}', 1) - \Delta T_f$ is a constant independent on N. Note that because the difference between N and N' is a small constant, $\Delta T_p - t_a(\mathbf{N}', 1)$ is also a constant. Therefore

$(\Delta T_p - \Delta T_f)$ is also a constant independence of N. This proves that when the optimal value of $\mathbf{T}/r$ is at the intersection of two constraints of the form (16), the periodic schedule is asymptotically time-optimal.

Case II: The second possibility is that the optimal solution is a vertex which happens to be on one of the axes: i.e. $\mathbf{T}/r$ is a solution of a system of the form

$$\begin{array}{l} \mathbf{T} \cdot \sum_c \delta(u,v) = r \cdot \sum_c d(u,v) \\ \mathbf{T} \cdot (1,0) = 0 \end{array}$$

(or $(0,1)$ instead of $(1,0)$). In this case $T_1 = 0$. Let $\mathbf{C} = (C_1, C_2)$ and $\mathbf{N} = (N_1, N_2)$. We can define $\alpha = \lfloor N_2/C_2 \rfloor$. Then let $\mathbf{X} = (X_1, X_2) = \mathbf{N} - \alpha \cdot \mathbf{C}$.

Next we define $\mathbf{N}' = (X_1, 0) + \alpha \cdot \mathbf{C} = \mathbf{N} - (0, X_2)$. This point $\mathbf{N}'$ is close to $\mathbf{N}$ since $X_2 < C_2$. Since $T_1 = 0$, points (0,0) and $(X_1, 0)$ are scheduled at the same time clock 0, the critical path is simply of the form $\alpha\mathbf{C}$, which starts from point $(X_1, 0)$ and ends at point $\mathbf{N}'$ (see Figure 7, and the periodic schedule is clearly optimal.

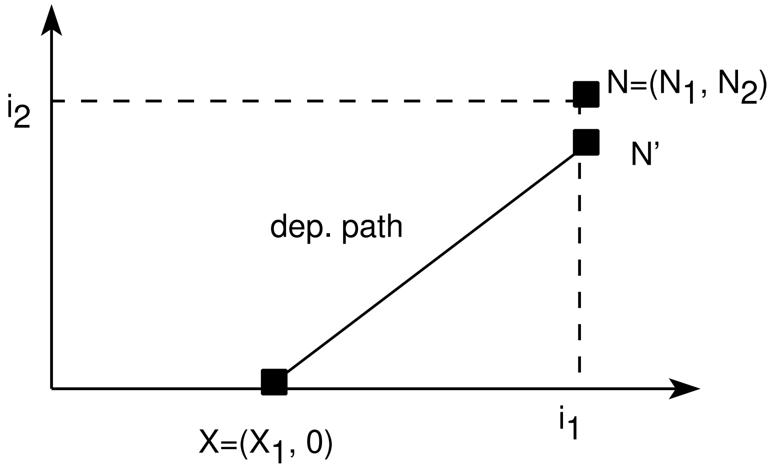


Figure 7: Dependence path from $(X_1, 0)$ to $\mathbf{N}'$.

■

5 Solution for Periodic Scheduling

In Section 5.1, we present an efficient algorithm to solve the periodic scheduling problem. In Section 5.2, we show how the periodic schedule can be used for loop transformation (or loop rewriting) so that maximum speedup can be obtained.

5.1 An Efficient Algorithm

The optimal value of $\mathbf{T}/r$ can be found by means of linear programming with the simplex method for example. The number of variables is $m+n$, where m denotes the number of statements of the loop and n is the depth of the loop nest.

In practice, n is small (usually 1, 2 or 3). Therefore, in this section, we propose a solution method which is very efficient for small n. The key observation is that in the linear program (see Theorem 4.1), the optimal value of T/r in the linear constraint (10) is bounded by the maximum computation rate of the so-called *critical cycles* in the dependence graph. Such bound can be derived efficiently by solving a generalized version of the so-called *minimum cost-to-time cycle ratio problem* [29]. This generalization has been inspired by our previous work for solving the one-dimensional problem [43]. The technique is first developed in this section for the case when r is unbounded. In Section 6, we generalize it to the case when r is fixed.

The basic strategy of our solution method for dimension n problem is to systematically reduce it to problems of lower dimensional problems and solve them accordingly. Without loss of generality, let us present the method for the case $n = 2$. An extension to cases for n greater than 2 is left for an exercise.

Let δ be the largest value of $\delta(u, v)$ and let d be the largest value of $d(u, v)$, that is,

$$\delta = \max_{(u,v)\in DG} \{\delta(u, v)\},$$

$$d = \max_{(u,v)\in DG} \{d(u, v)\}.$$

Given a simple loop as defined in Section 3 (with $n = 1$), let us first introduce the main results of [43] for computing the optimal $\mathbf{T}/r = T/r$ and the overall solution. (Reminder: we use lower case letter T to denote the fact that it is a scalar value).

1. The optimal value of T/r is in $[0, md]$.

2. The minimum distance between two distincts values of T/r is $1/(m\delta)^2$. The value of r is in $[1, m\delta]$.

3. One can test that there exists a particular schedule of a given T/r by solving a shortest path problem, which will in fact give valid values for the $A(v)$'s. When using Bellman-Ford algorithm, the complexity for doing so is in $\mathcal{O}(m^2)$. Using Gabow-Tarjan's algorithm, it is in $\mathcal{O}(m \lg m \sqrt{m})$. (Here, we consider that the number of dependences per statement is constant.)

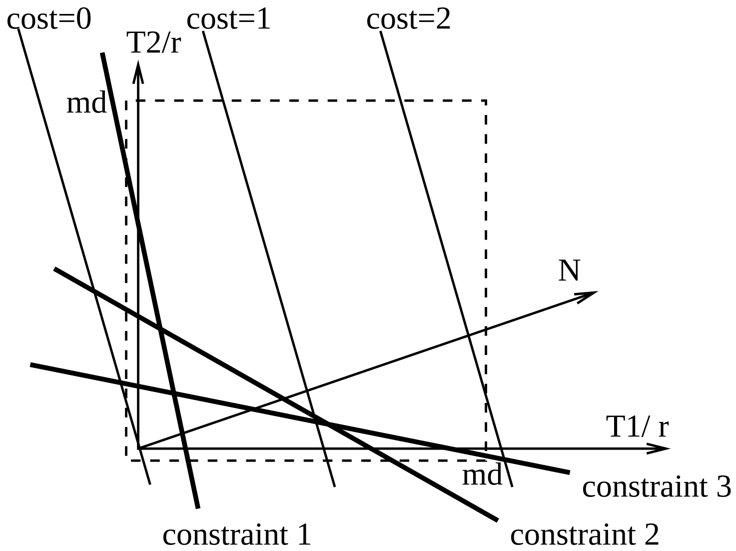
4. There are maximum $md(m\delta)^2$ distinct values for T/r to test. A binary search can be used to find the optimal value of T/r. After $(3 \lg m + \lg d + 2 \lg \delta)$ shortest path problems, the distance to the optimal value is smaller than the minimal distance between two distinct values of T/r. Let T'/r be this near-optimal value.

5. The optimal value can be found from this near-optimal solution by solving one additional shortest path which will detect the critical cycle of the DG. By using $T/r = T'/r - 1/(m\delta)^2$, one detects a negative cycle which is the critical cycle.

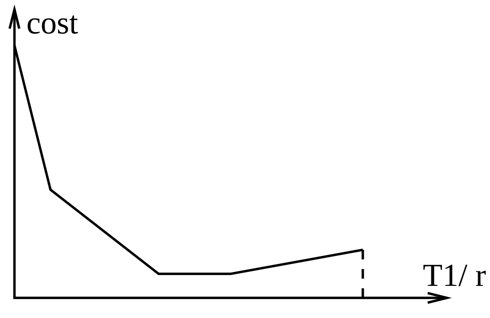
6. Given the critical cycle c, the optimal value of T/r can be found by solving $T/r = \sum_c d(u,v)/\sum_c \delta(u,v)$.

7. Valid $A(v)$'s can finally be found by means of one additional shortest path problem using the optimal value of T/r.

The complexity of this method to compute T/r, when using Bellman-Ford, is therefore in $\mathcal{O}(m^2(3\lg m + \lg d + 2\lg\delta + 1)) = \mathcal{O}(m^2 \lg m)$.



(a)



(b)

Figure 8: The basis for extending MCTRC to the case when $n = 2$.

We now generalize this to the case $n = 2$ to compute the optimal $\mathbf{T}/r$. As in the one dimensional case, our new algorithm performs a search of the optimal value of $\mathbf{T}/r$ through a bounded solution space. However, $\mathbf{T}/r$ is now a vector and the solution space is a two-dimensional region as illustrated in Figure 8. Consider Figure 8(a) first. The two axes represent T_1/r and T_2/r — the two components of $\mathbf{T}/r$. As explained in [19], there is one linear constraint per simple cycle C of the DG which is of the form:

$$\mathbf{T}/r \cdot \delta_c \geq d_c,$$

where δ_c is the summation of the dependence vectors along C and d_c is the sum of the associated delays. For simplicity, only three such constraints on $\mathbf{T}/r$ are shown in the figure. Note that the length of a cycle is at most m and the maximum value of a delay is d, $d_c \in [0, md]$. Furthermore, all components of δ_c are positive. Therefore each constraint cuts the axes T_1/r and T_2/r between 0 and md. The optimal value of $\mathbf{T}/r$ is at the intersection of two constraints and is therefore always within the dashed box shown in Figure 8(a).

The cost to minimize is $\mathbf{T}/r \cdot \mathbf{N}$ — a linear function which increases along $\mathbf{N}$. As shown in the figure, we use a set of parallel lines perpendicular to $\mathbf{N}$ to represent the solution points for $\mathbf{T}/r$ with equal cost.

Consider Figure 8(b) now. The axes of this diagram are T_1/r and the minimum "cost" of $\mathbf{T}/r$ for a particular value of T_1/r. Clearly, this cost is unimodal: it decreases first up to the point where it reaches its minimum and then increases. One key idea of the new algorithm is to perform a Fibonacci search [14] along T_1/r to find the optimal solution. At each point in the Fibonacci search, we need to compute the minimal cost for a given value of T_1/r and the corresponding value of T_2/r. As we will show below, this subproblem can be transformed and solved efficiently using the method developed for the case $n = 1$, (i.e. a binary search to solve the MCTRC problem).

Let us now explain how the binary search for a 2D problem is transformed into the 1-D problem. When $n = 2$, the constraints are of the form:

$$T_1\delta_1(u,v) + T_2\delta_2(u,v) + A(v) - A(u) \geq rd(u,v), \tag{27}$$

where $d(u,v)$ represents the delay for the dependence (u,v). We want to transform this constraint into the familiar 1-D form:

$$T_2\delta_2(u,v) + A(v) - A(u) \geq rd'(u,v).$$

By doing so, we can solve T_2/r by means of MCTRC. Let us see how this can be done. Clearly (27) can be rewritten as:

$$T_2\delta_2(u,v) + A(v) - A(u) \geq rd(u,v) - T_1\delta_1(u,v) = r(d(u,v) - T_1/r\delta_1(u,v)). \tag{28}$$

The problem is that we want $d'(u,v)$ to be an integer constant. In order to obtain this, we do variable substitution:

$$r_0 = r/r_1,$$

and

$$T_1' = T_1/r_0.$$

Then we obtain:

$$T_1/r = T_1/(r_0 \cdot r_1) = T_1'/r_1 = r_0 T_1'/(r_0 \cdot r_1).$$

By doing so, (28) becomes:

$$T_2\delta_2(u,v) + A(v) - A(u) \geq r_0(r_1 d(u,v) - T_1'\delta_1(u,v)).$$

That is:

$$d'(u,v) = r_1 d(u,v) - T_1'\delta_1(u,v). \tag{29}$$

The algorithm MCTRC performs a binary search over the range $[0, md]$, where d is the maximal delay. In (29), d' is function of r_1, but clearly $d' \leq r_1 d$. So now the binary search can be performed over $[0, mr_1 d]$.

Knowing how to solve the subproblem, we know return to the Fibonacci search procedure. The Fibonacci search will be along T_1'/r_1, and for each fixed value of T_1'/r_1, an optimal value of T_2/r_0 will be computed by means of MCTRC. The corresponding value of $\mathbf{T}/r$ will then be $(r_0 T_1', T_2)/(r_0 \cdot r_1)$.

How long the Fibonacci search will take to terminate and return the optimal solution ? ¿From theory of linear programming, the optimal $\mathbf{T}/r$ is at a vertex of the convex polyhedron defined by the set of linear constraints. As illustrated in Section 4.3, such a vertex is the solution a system of two equations of the form (22) – one for each cycle. Because r is at most the determinant of the matrix formed by the coefficients of the two equations, the value of r is in $[1, (m\delta)^2]$. (The coefficients are in $[0, m\delta]$.) We want to do a Fibonacci search along T_1'/r_1 up to a point when the interval contains only one vertex. Because of the maximal value of r, the minimum distance between two distinct vertices is $(m\delta)^4$. This tells us how far we must go in the Fibonacci search. It also tells us how large can r_1 be and the complexity of the associated MCTRC.

At the end of the Fibonacci search, one is left with a very small interval for T_1'/r_1 within which the optimal solution resides. At the left of this interval, MCTRC detects a critical cycle, say C_1; and at the right of this interval, MCTRC detects another critical cycle, say C_2. The optimal $\mathbf{T}/r$ is at the intersection of these two constraints. So it can be computed by solving the corresponding system of equations.

Here is a summary of the basic outline of our algorithm.

Algorithm 5.1

1. **Step I**: Perform a Fibonacci search on T_1'/r_1 in the range $[0, md]$ until the size of the interval is smaller that $1/(m\delta)^4$. The cost function to minimize is $(T_1'/r_1, T_2/(r_0 \cdot r_1)) \cdot (N_1, N_2)$. For each particular value of T_1'/r_1 do the following.
   (a) Transform the problem into the lower dimension.
   (b) Perform MCTRC to compute T_2/r_0 in the range $[0, r_1 md]$.
2. **Step II**: Find the final exact solution:
   (a) Detect the critical cycles defining the optimal solution.
   (b) Find the exact solution of $\mathbf{T}/r$ with the critical cycles.

$\circ$

The next theorem states the complexity of the algorithm.

Theorem 5.1 *The complexity of Algorithm 5.1 is*

$$\mathcal{O}(5 \lg m + \lg d + 4 \lg \delta) \cdot (m^2(7 \lg m + \lg d + 6 \lg \delta + 1)) = \mathcal{O}(m^2(\lg m)^2). \qquad (30)$$

Proof There are $md(m\delta)^4$ values of T_1/r_1 to consider. Thus, there are $\lg(md(m\delta)^4) = (5 \lg m + \lg d + 4 \lg \delta)$ iterations for the Fibonacci search of T_1'/r_1. For each value of T_1'/r_1, a particular MCTRC has to be performed over the range $[0, r_1 md]$. The complexity, when using Bellman-Ford to solve the shortest path problem, is in $\mathcal{O}(m^2(\lg r_1 + 3 \lg m + \lg d + 2 \lg \delta + 1))$. Because $r_1 \in [0, (m\delta)^4]$, it is in the worst case in $\mathcal{O}(m^2(7 \lg m + \lg d + 6 \lg \delta + 1)) = \mathcal{O}(m^2(\lg m)^2)$. Thus, the complexity of the overall algorithm is in (30).

$\blacksquare$

Let us end this section with a brief generalization of this algorithm to the case when $n > 2$. The volume of the search space is md^n, and the value of r becomes exponential with n. One can recursively use a Fibonnacci search along $n-1$ dimensions and still use the shortest path algorithm along the last dimension. The overall complexity is exponential with n. However, for small values of n (say $n < 5$), it is still very efficient. For $n = 3$, the complexity is basically in $\mathcal{O}(m^2(\lg m)^3)$.

5.2 Loop Rewriting

Given a loop and a feasible periodic schedule, this section illustrate how can the loop be rewritten efficiently to implement the schedule. This procedure involves the following steps:

1. Write the given periodic schedule in the form:

$$t(\mathbf{i}, v) = \frac{g\mathbf{T}' \cdot \mathbf{i} + A(v)}{r}$$

   in such a way that g and r are relatively prime non-negative integers and that the components of $\mathbf{T}'$ are relatively prime.

2. With $\mathbf{T}'$, build a unimodular transformation $\mathbf{U}$ whose first row is $\mathbf{T}'$. As explained in [16], this can be done by computing the Hermite form of $(\mathbf{T}')^{\mathrm{T}} = (1, 0, ...) \cdot \mathbf{U}$.

3. Rewrite the loop using the unimodular transformation $\mathbf{U}$.

4. In terms of $\mathbf{i}'$, the schedule is:

$$t(\mathbf{i}, v) = \frac{g\mathbf{T}' \cdot \mathbf{i} + A(v)}{r} = \frac{g(1, 0, ...) \cdot \mathbf{U} \cdot \mathbf{U}^{-1} \cdot \mathbf{i}' + A(v)}{r} = \frac{g(1, 0, ...) \cdot \mathbf{i}' + A(v)}{r}.$$

   In other words, the outer loop is now sequential and all the inner loops are parallel ("forall"). The loop rewriting becomes similar to the 1-dimensional case except that there are "forall" statements to schedule instead of single statements. Here, r "forall" statements are to be performed in g time steps. This can be achieved by "unrolling" the outer loop by a factor r, and by reordering the "forall" statements in the outermost loop according to the values of $A(v)$. (Loop rewriting for the 1-D case is explained in [42].)

The following example illustrates the rewriting procedure.

Example 2.3 (cont'd) The best schedule for loop (6) is:

$$t((i, j), v) = \lfloor \frac{3i + 2j + A(v)}{2} \rfloor = \lfloor \frac{1 \cdot (3, 2) \cdot (i, j) + A(v)}{r} \rfloor,$$

with $A(1) = 0$, $A(2) = 2$ and $A(3) = 1$. Here, $\mathbf{T}' = (3, 2)$, $g = 1$ and $r = 2$.

With $\mathbf{T}'$, we can now compute $\mathbf{U}$. For example, a possible value for $\mathbf{U}$ is

$$\mathbf{U} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}.$$

Using such a unimodular transformation, the loop becomes:

for $t_0 = 0$ to $5N$
  forall $p = pmin(t_0)$ to $pmax(t_0)$
    begin
     $i = t_0 - 2p || j = -t_0 + 3p;$
     $a(i,j) = a(i-1,j) + c(i-1,j);$
     $c(i,j) = b(i-1,j) + c(i-1,j);$
     $b(i,j) = a(i,j) + b(i,j-1);$
    end

This loop must be computed with a rate of 2 "forall" statements at every time step. This is achieved by rewriting it as follows.

for $t = 0$ to $\lfloor \frac{5N}{2} \rfloor + 1$
 $t1 = t - 1;$
   in parallel
    begin
      forall $p = pmin_1(t)$ to $pmax_1(t)$
        begin
         $i = 2t - 2p || j = -2t + 3p;$
         $a(i,j) = a(i-1,j) + c(i-1,j) ||$
         $c(i,j) = b(i-1,j) + c(i-1,j);$
        end
      forall $p = pmin_2(t)$ to $pmax_2(t)$
        begin
         $i = 2t + 1 - 2p || j = -(2t+1) + 3p;$
         $a(i,j) = a(i-1,j) + c(i-1,j);$
        end
      forall $p = pmin_1(t1)$ to $pmax_1(t1)$
        begin
         $i = 2t1 - 2p || j = -2t1 + 3p;$
         $b(i,j) = a(i,j) + b(i,j-1);$
        end
      forall $p = pmin_2(t1)$ to $pmax_2(t1)$
        begin
         $i = 2t1 + 1 - 2p || j = -(2t1+1) + 3p;$
         $c(i,j) = b(i-1,j) + c(i-1,j) ||$
         $b(i,j) = a(i,j) + b(i,j-1);$
        end
    end

□

As a remark, in the above example, we assume that each operation takes a unit of time. When operation delays are greater one, the rewriting procedure may depend on the architecture model used (e.g. can the operations be pipelined, etc.).

6 Periodic Scheduling with Given Periodicity

Often it may be desirable to use a periodic schedule with a fixed given periodicity. There are several pragmatic, yet important, reasons that a small given periodicity r is chosen for the scheduling. First, the code size of the result loop after rewriting is directly dependent on r, as illustrated in the previous section. Large code size may increase the program storage requirement, and degrades the effectiveness of the instruction cache. Secondly, large code size may also tend to increase the *register pressure* and affects the overall performance. Unfortunately, the optimal value r derived using methods described in the previous sections can become very large. This is illustrated by the following example.

Example 6.1 Consider a 3-nested loop whose associated DG is shown in Figure 9.

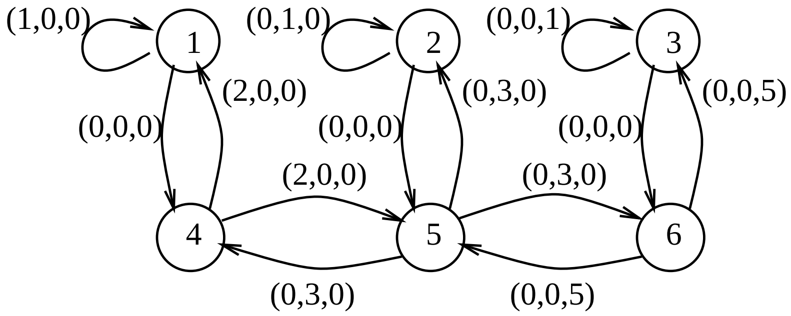


Figure 9: Dependence graph that leads to an irregular schedule.

The statements S_1, S_2 and S_3 take one unit of time, and S_4, S_5 and S_6 take respectively 2, 3 and 5 units of time to get executed. It can be shown that the best value of $\mathbf{T}/r$ is then:

$$(\frac{3}{2}, \frac{4}{3}, \frac{6}{5}) = \frac{(45, 40, 36)}{30},$$

in which $r = 30$.

It is easy to generalize this loop to the n-D case, so that r becomes exponential.

$\square$

Therefore, to avoid code explosion, we may need to choose a small r to begin with. In this section, we discuss how to derive a periodic schedule with a fixed periodicity.

The optimal r-periodic schedule for a given r can also be solve by using the formulation given in Theorem 4.1. Unfortunately, the problem is now an integer programming problem and is generally much harder than ordinary linear programming problems. In Section 6.1, we first show that the general problem of optimal periodic scheduling with a fixed periodicity is NP-complete. In Section 6.2, we present an efficient solution based on an extension of the MCTRC method similar to what was illustrated in Section 5.1.

6.1 NP-completeness Proof

In this section we show the optimal periodic scheduling with a fixed periodicity is NP-complete. First let us state the decision version of the optimization problem which asks for a "Yes" or "No" answer.

Given: A dependence graph G=(N, A, m, d), representing a n-dimensional nested loop, where N is the set of nodes, A the set of arcs,

$$\delta = \{\delta(u,v) = (\delta_1(u,v), \delta_2(u,v), \cdots, \delta_n(u,v)), (u,v) \in A\}$$

is the set of dependence distance vectors, and $d = \{d(u,v), (u,v) \in A\}$ is the set of delays on the arcs. Also given are a optimization direction vector $N = (N_1, N_2, \cdots, N_n)$ and positive integers r and U.

Question: Does there exist a vector $T = (T_1, T_2, \cdots, T_n)$, such that they satisfy all the constraints:

$$T_1 \sum_{(u,v)\in C} \delta_1(u,v) + T_2 \sum_{(u,v)\in C} \delta_2(u,v) + \cdots + T_n \sum_{(u,v)\in C} \delta_n(u,v)$$

$$\geq r \sum_{(u,v)\in C} d(u,v), \forall C \in \mathcal{C}(G) \tag{31}$$

$$T_u \geq 0, T_u \text{ integers}, \forall h \in N \tag{32}$$

and such that the objective function has a value at most U:

$$N_1T_1 + N_2T_2 + \cdots + N_nT_n \leq U \tag{33}$$

Theorem 6.1 *The Knapsack problem [1] can be reduced to the n-D MCTRC problem. Therefore the n-D MCTRC problem is NP-complete.*

Proof Actually we will prove a much stronger result that even restricting the n-D MCTRC to be the case that it contains one instruction in the loop body and only one dependence cycle may occur. But first we give the definition the Knapsack problem so that we can construct an instance for the n-D MCTRC problem.

Knapsack Problem:

Given: Non-negative integer vectors in n-D space:

$$c = (c_1, c_2, \cdots, c_n),$$
$$a = (a_1, a_2, \cdots, a_n),$$

and positive integers B and V.

Question: Does there exist a non-negative integer vector $x = (x_1, x_2, \cdots, x_n)$ in n-D space such that x satisfies the volume bound requirement:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq B, \tag{34}$$

and such that the value of x is at least V:

$$c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \geq V. \tag{35}$$

From this given instance of Knapsack Problem we will construct an instance for the n-D MCTRC problem. The instance has only one instruction in its loop body and it has only one dependence cycle:

- The node is called S.
- The only dependence is an arc from S to itself: (S, S).
- The dependence distance vector of (S, S) is:
$$(c_1, c_2, \cdots, c_n).$$
- The delay of S is defined to be V.
- Optimization direction vector is defined to be
$$N = (a_1, a_2, \cdots, a_n).$$
- integer r is fixed to be 1.

For this single cycle instance of n-D MCTRC problem its decision problem is the following form:

$$a_1 T_1 + a_2 T_2 + \cdots + a_n T_n \leq B, \tag{36}$$

and

$$c_1 T_1 + c_2 T_2 + \cdots + c_n T_n \geq V, \tag{37}$$

where (36) should be understood as the bound on the optimization direction, and (37) should be understood as the timing constraint. Now (36) and (37) together are exactly the inequalities in the decision problem of the Knapsack Problem with the variable name being changed to T. Therefore the Knapsack Problem can be solved if and only if the n-D MCTRC problem can be solved. This finishes the proof.

∎

6.2 An Efficient Solution Method

In the previous section, we have shown that the problem of computing the best periodic schedule with a fixed r is NP-complete. In this section, we present an efficient heuristic solution method for the case when the loop depth n is small. The algorithm is similar to the one presented for finding the asymptotically time-optimal periodic schedule (i.e. without r fixed). The only difference is that now the Fibonacci search is on an integer grid and not on a continuous interval.

As explained in [43], when $n = 1$, it is very simple to compute the best r-periodic schedule with r fixed. One can first compute the best near-optimal periodic schedule; and derive the optimal T/r ratio, say T_{opt}/r_{opt}. The best value of T is then given by $T = \lceil rT_{opt}/r_{opt} \rceil$.

Let us now consider the case when $n = 2$. We leave the reader with its generalization to the case when $n > 2$. For the case $n = 2$, we can slightly modify the algorithm for r-periodic scheduling given in subsection 5.1. The new algorithm for the case when $r = 1$ is given hereafter. It is straightforward to generalize it to the case when $r > 1$.

The algorithm of subsection 5.1 looks for T_1/r_1 and T_2/r_0 so that $\mathbf{T}/r$ is $(r_0T_1, T_2)/(r_0 \cdot r_1)$. When $r = 1$, we want the best $\mathbf{T}/1 = \mathbf{T}$. This is found by fixing r_1 to one, and applying the same algorithm. Thus, the Fibonacci search for finding the best T_1/r_1 is now looking only at integer points, to find the best value of $T_1/1$. Combined with this, the MCTRC method is used as before to find the best value of T_2/r_0. The best value of $\mathbf{T}$ is then $(T_1, \lceil T_2/r_0 \rceil)$. The values of $A(v)$'s are found as before by solving a particular shortest path problem.

Because r_1 is fixed to one, there are only md $(+1)$ values of $T_1/r_1 = T_1$ in the interval $[0, md]$ to consider instead of $md(m\delta)^4$. Thus, the complexity of this new algorithm is now

$$\mathcal{O}(m^2(5 \lg m + \lg d + 4 \lg \delta + 2)(\lg m + \lg d + 4 \lg \delta + 2)) = \mathcal{O}(m^2(\lg m)^2).$$

This is slightly better than for near-optimal periodic scheduling.

7 A Discussion of Related and Future Work

Machines built on advanced VLSI microprocessor architectures must take advantage of the ample resources provided by the hardware to aggressively exploit fine-grain parallelism in nested loops: both within a uniprocessor and between multiple processors. Software pipelining has been successfully applied as a fine-grain loop scheduling method in compilers for high-performance pipelined processor architectures, as well as multiple-issue architectures such as super-scalar and VLIW architectures [2, 3, 4, 20, 21, 27, 32, 36, 40]. However, most of these existing methods only handle single-level loops.

Unimodular transformation has been proposed as a method to exploit parallelism in nested loops for parallelizing compilers [7, 44] and for systolic array design (see for example [10, 12]). It provides a framework to combine other loop transformation techniques such as *loop interchange*, *loop skewing* and *loop reversal*. However, some existing unimodular transformation treats the loop body (with multiple statements) as a single unit in the transformation. Furthermore, in most compiler oriented work on unimodular loop transformation, the focus has not been on loop scheduling and, as a result, statement timing has not been considered explicitly as a parameter in the problem formulation.

In Section 2, we demonstrated that simply performing unimodular transformation (according only to the dependence structure) followed by an instruction scheduling (considering instruction timing) may not be enough to obtain the best possible schedule for a loop. We have argued that it is essential to formulate the time-optimal scheduling problem in a unified framework where the dependence structure as well as statement timing constraints can be fully represented. The r-periodic scheduling scheme proposed in this paper is designed to provide such a framework.

Cyclic scheduling problems, a class of task scheduling problems where tasks are only constrainted by uniform precedence relations, have been studied by Chretienne and his colleagues [13, 11]. Chretienne has proved that the earliest schedule (free schedule) is optimal and that the optimal asymptotic behavior is r-periodic. Recently, these results have been applied in a number of applications. But their model is yet to be applied to nested loops.

Various forms of periodic scheduling method has also been used in the area of systolic array design. See for example [19, 31, 19, 35, 26, 25]. Some scheduling techniques have explicitly taken into account the execution delays of the individual statements by using periodic schedules [35, 26, 25]. Our model is based on this work.

Recently, there have been growing interests in applying periodic scheduling method in the compilation of loops. In [43], Van Dongen, Gao and Ning have suggested that time-optimal scheduling for one-dimensional loops can be formulated as a r-periodic scheduling problem. An efficient polynomial time algorithm has been presented to solve it based on the MCTRC method. The same technique was developed independently and presented at the same time in [24]. In [33], Ning and Gao have applied the periodic scheduling method to solve register allocation problem for time-optimal software pipelining. The work along this line was in part inspired by the pioneer work of Reiter on scheduling of Karp-Miller style computation graphs [38].

This paper is among the first to extend the r-periodic scheduling presented in [43] to nested loops. To do so, we have restricted ourselves to nested loops defined on a rectangular domain, whose dependence vectors have non-negative coefficients and whose DG is strongly connected. For this class of loops, we have proven that r-periodic periodic schedules are asymptotically optimal and that efficient solutions can be developed based on extensions of the MCTRC solution method. The model can easily be extended to loops whose DG is not strongly connected. In such cases, a *loop distribution* can be performed to separate a loop into a set of loops with strongly connected DGs. A more challenging generalization is when some components of the dependence vectors are negative, and the domain is a convex polyhedron instead of rectangular. In general, one can first transform the loop (e.g. using unimodular transformation) such that the dependence vectors become non-negative [27]. If the transformed loop is defined on a rectangular domain, one can successfully apply our search technique developed in previous sections. However, the result index domain may no longer be rectangular. Optimization techniques based on linear programming were proposed in [26, 25] to find the optimal schedule. But the number of problems to solve can be quite large. One open problem is to prove that periodic schedules are still asymptotically optimal, and that the MCTRC algorithm can remain a basic building block to efficiently find the optimal schedule. We leave this as future work.

In [15], periodic schedules are proven to be asymptotically optimal when the loop contains one statement only and the domain index is a sufficiently "fat" polyhedron. In [39], periodic scheduling was used for nested loops and it was demonstrated that it combines three previously known loop transformations: a unimodular change of basis, the *index shift method* [30], and cycle shrinking [34]. By means of an example, it was also proved that the gain of parallelism from the one obtained with unimodular transformations only may become unbounded.

In [16, 17, 18], Darte, Risset, and Robert have proposed a periodic scheduling scheme, called *affine-by-statement loop scheduling*. A major feature is that each statement in the loop body may be associated with a different *affine schedule* (e.g. a different $\mathbf{T}/r$ for each statement). Compare with the r-periodic scheduling method used in this paper, the affine-by-statement scheduling framework can handle a more general class of loops, e.g. non-perfect loops, loops with arbitrary index domain, loops with more than one strongly connected component, and loops with negative components in their distance vectors. In [22], Feautrier has proposed another interesting scheme which may apply different scheduling functions to different parts of index domain if necessary. These piecewise-affine schedules have proven to be powerful enough to express the free schedule of regular dependence graphs defined on a cone [5]. Both affine-by-statement and piecewise-affine schedules are more powerful than the model proposed in this paper, and hence are more computationally expensive to solve. It remains to be demonstrated to what extent such added power can be efficiently utilized in compiling real programs.

8 Conclusions

The efficient exploitation of fine-grain parallelism in loops is a major challenge in the design of next-generation high-performance computers. This paper investigates the extension of *software pipelining* to nested loops. We have proposed to use r-periodic scheduling as a framework and derived a simple linear programming formulation of the problem as well as an efficient solution.

One important application is to extend this framework to study automatic data partitioning and distribution for SIMD/MIMD distributed memory machines. In particular, we plan to investigate a problem formulation for automatic data partitioning which permits a smooth integration of both data partitioning and periodic loop scheduling. This capability, essential with loops in which arrays are recursively defined by loop-carried dependences, may require the extension of the data parallel paradigm.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Co., 1974.

[2] A. Aiken. Compaction-based parallelization. (PhD thesis), Technical Report 88–922, Cornell University, 1988.

[3] A. Aiken and A. Nicolau. Optimal loop parallelization. In *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, Atlanta, Georgia, June 22–24, 1988. ACM SIGPLAN. Also in *SIGPLAN Notices*, 23(7), July 1988.

[4] A. Aiken and A. Nicolau. A realistic resource-constrained software pipelining algorithm. In *Proceedings of the Third Workshop on Programming Languages and Compilers for Parallel Computing*, Irvine, CA, August 1990.

[5] W. Backes, U. Schwiegelshohn, and L. Thiele. Analysis of free schedule in periodic graphs. In *SPAA'92*, pages 333–339. ACM, 1992.

[6] U. Banerjee. *Dependence Analysis for Supercomputing*. Kluwer Academic Publishers, Boston, Massachusetts, 1988.

[7] U. Banerjee. Unimodular transformations of double loops. In *Proceedings of the Third Workshop on Programming Languages and Compilers for Parallel Computing*, Irvine, CA, August 1990. Also published in *Monographs in Parallel and Distributed Computing*, Pages 192–219, Pitman, 1991.

[8] U. Banerjee. Unimodular transformations of double loops. In *Advances in Languages and Compilers for Parallel Processing*, pages 192–219. MIT Press, Cambridge, Massachusetts, 1991.

[9] C.J. Brownhill and A. Nicolau. A hierarchical parallelizing compiler for VLIW/MIMD machines. In *Proceedings of the 5-th Workshop on Languages and Compilers for Parallel Computing*, August 1992.

[10] P.R. Cappello and K. Steiglitz. Unifying VLSI array design with linear transformations of space-time. In F.P. Preparata, editor, *Advances in Computing Research*, Greenwich, U.K., 1984. JAI Press Inc.

[11] J. Carlier and P. Chrétienne. *Problèmes d'ordonnancement: modélisation / complexité / algorithmes.* Etudes et recherches en informatique. Masson, Paris, 1988.

[12] M.C. Chen. Synthesizing systolic designs. In *1985 International Symposium on VLSI Technology, Systems and Applications*, 1985.

[13] P. Chretienne. *Les réseaux de Petri temporisés.* PhD thesis, Thèse d'état, Université P. et M. Curie, 1983.

[14] L. Cooper and D. Steinberg. *Introduction to methods of optimization.* W.B. Saunders Company, 1970.

[15] A. Darte, L. Khachiyan, and Y. Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):pp. 73–81, 1991.

[16] A. Darte, T. Risset, and Y. Robert. Loop nest scheduling and transformations. In *Proc. of Environments and Tools for Parallel Scientific Computing, J.J. Dongarra and B. Tourancheau eds., North Holland (1993)*, July 1992.

[17] A. Darte and Y. Robert. Affine-by-statement scheduling of uniform loop nests over parametric domains. Rapport 92-16, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, April 1992.

[18] A. Darte and Y. Robert. Scheduling uniform loop nests. Rapport 92-10, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, February 1992.

[19] J.M. Delosme and I.C.F. Ipsen. Efficient systolic arrays for the solution of Toeplitz systems: an illustration of a methodology for the construction of systolic architectures for VLSI. In *Systolic Arrays*, 1986.

[20] K. Ebcioğlu. A compilation technique for software pipelining of loops with conditional jumps. In *Proceedings of the 20th Annual Workshop on Microprogramming*, December 1987.

[21] K. Ebcioğlu and A. Nicolau. A global resource-constrained parallelization technique. In *Proceedings of the ACM SIGARCH International Conference on Supercomputing*, June 1989.

[22] P. Feautrier. A collection of papers on the systematic construction of parallel and distributed programs. Technical Report Hors-série, Lab. MASI, Université P. et M. Curie, 4 Place Jussieu 75252 Paris Cédex 05, June 1992.

[23] G. R. Gao, Y. B. Wong, and Qi Ning. A Petri-Net model for fine-grain loop scheduling. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 204–218, Toronto, Ontario, June 26–28, 1991. ACM SIGPLAN. Also in *SIGPLAN Notices*, 26(6), June 1991.

[24] F. Gasperoni. *Scheduling for Horizontal Systems: The VLIW Paradigm in Perspective.* PhD thesis, Courant Institute of Mathematical Sciences, New York University, 1991.

[25] H.V. Jagadish, S.K. Rao, and T. Kailath. Array architectures for iterative algorithms. *Proceedings of the IEEE*, 1987.

[26] B. Joinnault. Conception d'algorithmes et d'architectures systoliques. Thèse de l'Université de Rennes I, Sept 1987.

[27] Monica Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, pages 318–328, Atlanta, Georgia, June 22–24, 1988. ACM SIGPLAN. Also in *SIGPLAN Notices*, 23(7), July 1988.

[28] L. Lamport. The parallel execution of DO loops. *Communications of the ACM*, pages 83–93, February 1974.

[29] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Saunders College Publishing, Ft Worth, TX, 1976.

[30] L.S. Liu, C.W. Ho, and J.P. Sheu. On the parallelism of nested for-loops using index shift method. In *Proc. of International Conf. on Parallel Processing*, pages II–119–II–123, Aug. 1992.

[31] D.I. Moldovan. On the design of algorithms for VLSI systolic arrays. *IEEE Proceedings*, Vol. 71(N. 1), January 1983.

[32] A. Nicolau, K. Pingali, and A. Aiken. Fine-grain compilation for pipelined machines. Technical Report TR-88-934, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[33] Q. Ning and G.R. Gao. A novel framework of register allocation for software pipelining. In *to appear in POPL '93*, Charleston, South Carolina, January 11–13 1993.

[34] Constantine Polychronopoulos. Toward auto-scheduling compilers. Technical report, University of Illinois at Urbana-Champaign–Center for Supercomputing Research and Development, May 1988. CSRD Rpt. No. 789.

[35] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc. IEEE 11-th Int. Sym. on Computer Architecture*, 1984.

[36] B. R. Rau and C. D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proceedings of the 14th Annual Workshop on Microprogramming*, pages 183–198, 1981.

[37] B.R. Rau, M.S. Schlansker, and P.P. Tirumalai. Code generation schema for modulo scheduled loops. In *Proceedings of the 25-th International Symposium on Microarchitectures, Portland, Oregon*, 1992.

[38] Raymond Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, 1968.

[39] Y. Robert and S. Song. Revisiting cycle shrinking. *Parallel Computing*, pages pp. 481–496, 1992.

[40] R. F. Touzeau. A FORTRAN compiler for the FPS-164 scientific computer. In *Proceedings of the SIGPLAN '84 Symposium on Compiler Construction*, pages 48–57, Montréal, Québec, June 17–22, 1984. ACM SIGPLAN. Also in *SIGPLAN Notices*, 19(6), June 1984.

[41] V. Van Dongen. *From systolic to periodic array design.* PhD thesis, Thèse de l'Université Catholique de Louvain, January 1991.

[42] V. Van Dongen and G. Gao. A polynomial time method for optimal software pipelining. Technical Report ACAPS Tech. Memo 37, McGill University, School of Computer Science, 1992.

[43] V. Van Dongen, G. Gao, and Q. Ning. A polynomial time method for optimal software pipelining. In *Proceedings of CONPAR '92*, Lecture Notes in Computer Science 634, Paris, France, September 1992.

[44] Michael E. Wolf and Monica S. Lam. A data locality optimizing algorithm. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30–44, Toronto, Ontario, June 26–28, 1991. ACM SIGPLAN. Also in *SIGPLAN Notices*, 26(6), June 1991.

[45] Michael J. Wolfe. *Optimizing Supercompilers for Supercomputers.* Pitman, London and MIT Press, Cambridge, MA, 1989. In the series, Research Monographs in Parallel and Distributed Computing. Revised version of the author's Ph.D. dissertation, Published as Technical Report UIUCDCS-R-82-1105, University of Illinois at Urbana-Champaign, 1982.