Revisiting cycle shrinking

Yves Robert*

Laboratoire de l'Informatique du Parallélisme - IMAG

École Normale Supérieure de Lyon

46, Allée d'Italie

69364 Lyon Cédex 07 - France

e-mail: yrobert@lip.ens-lyon.fr

Siang W. Song†

University of São Paulo

Institute of Mathematics and Statistics

Department of Computer Science

C.P. 20570 - São Paulo, SP 01498 - Brazil

e-mail: song@brusp.ansp.br

Abstract

Several loop transformations techniques have been designed to extract parallelism from nested loop structures. We first review two important approaches, known as *Generalized Cycle Shrinking* presented by Shang, O'Keefe and Fortes and *the Index Shift Method* introduced by Liu, Ho and Sheu. The main result of the paper is a new methodology that permits to combine cycle shrinking techniques with the index shift method. We present a new optimization method that produces the best scheduling vector, and we show that we can outperform previous results by an arbitrary speedup factor.

Key-words: cycle shrinking, nest of loops, scheduling vector, retiming, index shift, partitioning technique, uniform recurrence equations.

*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. Part of this work has been done while the first author was visiting IME/USP in July 1991. Support from the BID/USP is gratefully acknowledged.

†Supported by the Program BID/USP (Interamerican Development Bank/University of São Paulo) - Project 30.01, FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) - Proc. No. 88/3095-9 and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) - Proc. No. 306063/88-3.

1 Introduction

Several loop transformations techniques have been designed to extract parallelism from nested loop structures [1, 2, 7, 14, 15]. As pointed out by Polychronopoulos [6], by Wolfe [14] or by Shang, O'Keefe and Fortes [12, 13], this task is often performed by optimizing and parallelizing compilers that have as their goal the transformation and mapping of a serial program into a parallel form that can be executed on a particular architecture. Nested loop structures offer the most fruitful sources of parallelism in serial programs, and it is therefore of paramount importance that the analysis necessary for such programs be both precise and efficient.

In [6], three loop transformation techniques - *simple cycle shrinking, selective cycle shrinking* and *true dependence cycle shrinking* - were introduced to transform sequential loops into parallel loops. Shang, O'Keefe and Fortes [12, 13] have recently improved these results along two directions:

- the three different transformations are put into a unified framework
- methods to derive optimal cycle shrinking are presented

Optimal cycle shrinkings are determined through an optimization procedure described in [11].

Another technique for improving the parallelism of loop structures has been introduced by Liu, Ho and Sheu [4] as the *Index Shift Method*. The idea behind this method is to defer or advance the execution steps of some statements such that the total execution time of loops can be reduced.

The main result of the paper is a new methodology that permits to combine cycle shrinking techniques with the index shift method. More precisely, the paper is organized as follows. In Section 2 we briefly review some background material concerning nests of loops and cycle shrinking, and we define some terminology taken from Polychronopoulos [6]. In Section 3 we discuss Generalized Selective Cycle Shrinking (GSS), a technique introduced by Shang, O'Keefe and Fortes [13]. In Section 4 we present the Index Shift Method (ISM) from Liu, Ho and Sheu [4], which can be viewed as a refinement of GSS. We show how to improve on the ISM method in this context. In Section 5 we combine both ideas. It turns out that the best results are not obtained through GSS followed by ISM or its modified version, but rather a modification of the optimization goal from the beginning. Throughout, we use simple examples borrowed from previous papers, so that the reader can retrieve familiar concepts and can more easily appreciate our improvements. We conclude Section 5 with an example showing that indeed these improvements can enable us to gain an order in performance. Conclusions are given in Section 6.

2 Background material

Throughout the paper, we consider perfect nests of loops of the following form:

GLN (General Loop Nest) class

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
{ statement S_1 }
{ statement S_2 }
$\vdots$
{ statement S_k }
end

Here, l_1 and u_1 are assumed to be constants, while $l_j(i_1, \ldots, i_{j-1})$ (resp: $u_j(i_1, \ldots, i_{j-1})$) is the minimum (resp: maximum) of a finite number of affine functions of $i_1, \ldots, i_{j-1}$. Also, we require that all variables instanciated in the statements S_1 to S_k be affine functions of the loop indices $i_1, i_2, \ldots, i_n$. The index set for the loop nest is defined as $Dom = \{I = (i_1, \ldots, i_n) \in Z^n, l_j \leq i_j \leq u_j \text{ for } 1 \leq j \leq n\}$. This framework seems to be pretty general and widely used [1, 2, 15].

Data dependences between instances of the statements $S_1, \ldots, S_k$ are defined according to Banerjee [1] or Polychronopoulos [6]. We write $S_u \delta S_v$ if there exist index values $(i_1, \ldots, i_n)$ and $(j_1, \ldots, j_n)$ in Dom such that

i. $(i_1, \ldots, i_n) \leq (j_1, \ldots, j_n)$ where $\leq$ is the lexicographic order over Z_n.

ii. statement $S_u(i_1, \ldots, i_n)$ must be executed before statement $S_v(j_1, \ldots, j_n)$ to preserve the semantics of the nest and we let $\delta = (j_1 - i_1, j_2 - i_2, \ldots, j_n - i_n)$ be the dependence vector between the two statements.

As pointed out in [6], dependences can be further divided into three categories (flow-, anti- or output-dependences). Also, given two statements S_u and S_v, there can be several index pairs (I, J) in Dom^2 for which $S_v(J)$ depends upon $S_u(I)$, hence several dependence vectors between S_u and S_v.

An important subclass of GLN is when all dependences between statements correspond to a fixed dependence vector, independent of the index pairs instanciating the dependence. Such nests of loops are termed *uniform*. Uniform nests of loops have a tremendous importance and have received considerable attention, and this for at least the following reasons:

- they arise naturally in many scientific applications,
- powerful mathematical tools can be used to deal with them, and
- all results for uniform nests can be applied to general nests if the conservative approach of retaining only the smallest dependence vector among all dependence vectors linking two statements is taken.

Consider the following example slightly modified from Polychronopoulos [6]:

Example 1:

for $i = 0$ to N_1 do
for $j = 0$ to N_2 do
begin
{ Statement S_1 } $a(i,j) = b(i-3, j-5)$
{ Statement S_2 } $b(i,j) = a(i-2, j-4)$
end

In this example, the index set is a rectangle

$$Dom = \{(i,j) \in Z^2, 0 \leq i \leq N_1, 0 \leq j \leq N_2\}.$$

Variable $a(i,j)$ is produced in statement $S_1(i,j)$ and consumed in statement $S_2(i-2, j-4)$ so there is a dependence between S_1 and S_2 of vector $(2,4)$. Similarly, there is a dependence between S_2 and S_1 of vector $(3,5)$. Both dependences are uniform, so we let

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

represent all the dependences. Note that all dependence vectors are lexicographically positive (their first non-zero component is greater than 0), due to the condition (i). The dependence graph is represented in Figure 1.

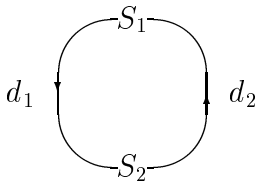

Figure 1: Dependence graph of Example 1

In this example, there is a dependence cycle, since there is a dependence between S_1 and S_2 and another one between S_2 and S_1. More generally, we say that there is a dependence cycle of length L if there are L different statements $S_{l_1}, S_{l_2}, \ldots, S_{l_L}$ such that $S_{l_1} \delta S_{l_2} \delta \ldots \delta S_{l_L} \delta S_{l_1}$. In our example the cycle is of length 2. Of course there can exist several cycles of different lengths involving the k statements S_1 to S_k.

In general, loops whose statements are involved in a dependence cycle are considered to be serial. However, techniques such as cycle shrinking can be used to extract parallelism that may still be present in the loop. We discuss such techniques in the next section.

Beforehand, we point out that most of the examples that we use hereafter will have the following simplified form (as the previous example):

RUN (Regular Uniform Nest) class

i. the index set is a parallelepiped

```
for i_1 = 0 to N_1 do
for i_2 = 0 to N_2 do
    ⋮
for i_n = 0 to N_n do
    begin
          { Statement S_1 }
                    ⋮
          { Statement S_k }
    end
```

ii. the dependences are uniform and all dependence vectors are recorded in the dependence matrix D of size $n \times m$, where m is the number of such vectors.

The restriction that the index domain is a parallelepiped is only for technical convenience. We will even use hypercubic domains where all N_i's are equal. Also, the fact that the loop indices range from 0 to N by increment 1 is no loss of generality. Such nests are called normalized by [6]. The main restriction is indeed that we deal with *uniform* nests.

3 Generalized Selective Cycle Shrinking

Generalized cycle shrinking has been introduced by Shang, O'Keefe and Fortes [13] as a generalization of a well-known technique in compiler optimization called selective shrinking [6]. Consider a GLN nest. The idea is to determine a vector $\Pi = (\pi_1, \ldots, \pi_n)$ such that all index points I, J in Dom such that $\Pi I = \Pi J$ can be executed simultaneously (assuming a target multiprocessor machine). Such a vector Π will be called a *scheduling vector*.

Of course not any vector can be used as a scheduling vector. The semantics of the loop has to be preserved. In [13], the following theorem is proven:

GSS theorem: Consider a GLN nest with uniform dependences, and let $D = (d_1, \ldots, d_m)$ be the $n \times m$ dependence matrix. Then any vector $\Pi = (\pi_1, \ldots, \pi_n)$ such that

$$\begin{array}{ll} (i) & \Pi D > 0 \\ (ii) & \gcd(\pi_1, \ldots, \pi_n) = 1 \end{array}$$

can be used as a scheduling vector. Any index I in Dom will be executed at time-step $\lfloor \frac{\Pi I}{\mathrm{disp}(\Pi)} \rfloor$, where $\mathrm{disp}(\Pi) = \min \{ \Pi d_j, 1 \leq j \leq m \}$ is the displacement.

The idea can be cast into the following terms: we group the index points into packets, and to execute the nest we have an outer sequential loop that represents time. All index points that are assigned the same time are executed concurrently. In other words, we have something like:

for $time$ = min_time(Dom) to max_time(Dom) do
    execute all index points I in Dom such that
        $\lfloor \frac{\Pi I}{\text{disp}(\Pi)} \rfloor = time$

Of course, min_time(Dom) and max_time(Dom) represent the smallest and largest values of the scheduling time for an index I in Dom.

Intuitively, condition (i) ensures that the semantics of the loop will be preserved. In other words the scheduling vector Π is safe if it forms an acute angle with the dependence vectors. Also, we see that the scheduling time for an index point is homogeneous, hence the restriction that the components of Π be relatively prime.

Now, the problem is to determine the best scheduling vector, that is a vector Π that satisfies (i) and (ii) and for which max_time(Dom) - min_time(Dom) is minimum. This last condition is clearly equivalent to minimizing

$$\frac{\max\{\Pi I - \Pi J, I, J \in Dom\}}{\text{disp}(\Pi)}.$$

Shang, O'Keefe and Fortes [13] propose a procedure that is able to determine the best scheduling vector in some cases, including the RUN case. The reason why they call their approach Generalized Selective Shrinking is the following: assume that there is a cycle between the statements S_1 to S_k of the nest. A technique from compiler optimization known as selective shrinking [6] is to consider the rows of the dependence matrix D one after the other, and to select the first one such that all entries are positive. The smallest of these entries can safely be chosen as the displacement. We see that it amounts to try successively the basis vectors as possible scheduling vectors.

Another technique from compiler optimization is known as true dependence shrinking [5, 6]. It consists in computing the true distance for each dependence vector d_i, defined to be the number of loop iterations separating the instanciation of the corresponding dependent statements.

For a RUN, we get

$$\text{true_distance}(d_i) = d_{n,i} + d_{n-1,i} * (N_n + 1) + d_{n-2,i} * (N_n + 1)(N_{n-1} + 1) + \ldots + d_{1,i} \prod_{j=2}^{n} (N_j + 1)$$

where

$$d_i = \begin{pmatrix} d_{1i} \\ d_{2i} \\ \vdots \\ d_{ni} \end{pmatrix}$$

and the displacement is the minimum of the true distances over all dependence vectors. This technique can be interpreted as proposing $\Pi = (\prod_{j=2}^{n}(N_j + 1), \prod_{j=3}^{n}(N_j + 1), \ldots, 1)$ for scheduling vector.

We conclude this section by stating the following problem.

GSS optimization problem: Find a vector Π with relatively prime components and such that $\Pi D > 0$ which minimizes

$$\mathrm{GSS}(\Pi) = \frac{\max\{\Pi I - \Pi J, I, J \in Dom\}}{\mathrm{disp}(\Pi)}.$$

Consider Example 1 again. Let $\Pi = (a, b)$ be a scheduling vector. We have $\Pi d_1 = 2a + 4b, \Pi d_2 = 3a + 5b$, hence $\mathrm{disp}(\Pi) = \min(2a + 4b, 3a + 5b)$ and max_time(Dom) - min_time(Dom) = $\mid a \mid N_1 + \mid b \mid N_2$. The problem is then to find two integers (a, b) such that the quantity

$$\frac{\mid a \mid N_1 + \mid b \mid N_2}{\min(2a + 4b, 3a + 5b)}$$

is minimized, subject to the constraints

$$\gcd(a, b) = 1, \quad 2a + 4b \geq 1, \quad 3a + 5b \geq 1.$$

Shang et al. use the method in [9] to find the best scheduling vector. They identify two possible solutions, $\Pi_1 = (1, 0)$ and $\Pi_2 = (0, 1)$. Π_1 leads to

$$\mathrm{GSS}(\Pi_1) = \frac{N_1}{2}$$

and Π_2 leads to

$$\mathrm{GSS}(\Pi_2) = \frac{N_2}{4}.$$

Depending upon the domain shape, either vector can be optimal.

4 The Index Shift Method

4.1 Description of the Index Shift Method

The Index Shift Method (ISM) has been introduced by Liu, Ho and Sheu [4]. It is best explained with an example, which we take from Peir and Cytron [5]:

Example 2:

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

The dependences are the following:

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

We obtain the following dependence matrix:

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

The index domain is a square:

$$Dom = \{(i,j) \in Z^2, 0 \leq i,j \leq N\}.$$

There are two dependence cycles, namely $C_1 = (S_1, S_3, S_2)$ and $C_2 = (S_1, S_4)$.

Let us first compute the best scheduling vector with the GSS approach. We search for $\Pi = (a,b)$ such that

$$\begin{aligned} &\gcd(a,b) = 1 \\ &\Pi D > 0 \Longleftrightarrow \begin{cases} b \geq 1 \\ a \geq 6b+1 \end{cases} \end{aligned}$$

Note that a and b are necessarily positive. We get $\text{disp}(\Pi) = \min(2b, a-6b, a+5b, b, a-4b)$ $= \min(a-6b, b)$ and we want to minimize the quantity

$$\text{GSS}(\Pi) = \frac{(a+b)N}{\text{disp}(\Pi)}.$$

The following straightforward case analysis shows that $\Pi = (7,1)$, for which $\text{GSS}(\Pi) = 8N$, is the best solution:

i. if $b = 1$, then minimize $\text{GSS}(\Pi) = (a+1)N$, hence take $a = 6b+1 = 7$

ii. if $b \geq 2$, then consider two subcases:

(a) if $6b+1 \leq a \leq 7b-1$, then $\text{disp}(\Pi) = a-6b$, and $\text{GSS}(\Pi) = \frac{a+b}{a-6b}N$, hence take $a = 7b-1$ and get $\text{GSS}(\Pi) = \frac{8b-1}{b-1}N \geq 8N$

(b) if $7b+1 \leq a$, then $\text{disp}(\Pi) = b$, and $\text{GSS}(\Pi) = \frac{a+b}{b}N$, hence take $a = 7b+1$ and get $\text{GSS}(\Pi) = \frac{8b+1}{b}N \geq 8N$. (Note that $a = 7b$ is excluded because $\gcd(7b, b) = b \geq 2$.)

Next, draw a directed graph (Figure 2) whose nodes are the statements S_1 to S_4 and whose edges are weighted by the values $e_i = \Pi d_i$.

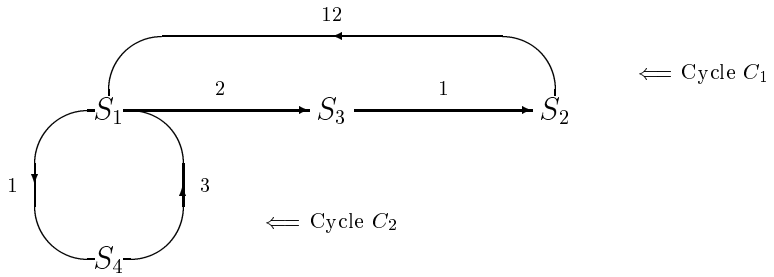

Figure 2: Dependence cycles of Example 2

The idea of the Index Shift Method is to apply some retiming to this graph. Indeed, consider cycle C_2 with the edge $e_{14} = (S_1, S_4)$ weighted $w_{14} = 1$ and the reverse edge $e_{41} = (S_4, S_1)$ weighted $w_{41} = 3$. Since $\text{disp}(\Pi)$ is the minimum of the weights of the edges in the graph, we would like to remove 1 from w_{41} and to add 1 to w_{14}. Similarly, $w_{21} = 12$ and we can shift part of this weight to w_{13} and w_{32}. Such shifts can be conducted so that the minimum weight of an edge in the graph becomes 2.

Systolicians will have recognized the retiming procedure of Leiserson and Saxe [3]. The idea is to modify weights by adding some given quantity (possibly negative) to all incoming edges to a given node, and to subtract the same quantity from all outgoing edges from that node. Obviously, such transformations do not modify the total weight of each cycle. Hence, if the goal is to achieve a good balance among all weights involved in a cycle of length k and total weight T, the best you can do is to assign the weight $\lfloor T/k \rfloor + 1$ to (T mod k) edges and the weight $\lfloor T/k \rfloor$ to the remaining ones.

Liu et al. [4] provide a method for determining the transformations needed to achieve this balancing of the weights among all cycles. The method is the same as in [8, page 252], and it is simpler than the one originally proposed in [3]. Let us denote by S_i^k the transformation that consists of adding k to the weights of all incoming edges to node S_i, and subtracting k from the weights of all outgoing edges from S_i. In our example, the (commutative) sequence $S_2^1 S_4^1$ leads to the graph of Figure 3.

The minimum weight is now 2, which means that the new displacement is 2.

What does it mean to apply a transformation S_i^k to the graph for the nest of loops, and how the range of the loop indices is affected? First of all, applying the transformation S_i^k amounts to replacing the execution of a given instance $S_i(J_1)$ of statement S_i by the execution of another instance $S_i(J_2)$ such that the scheduling difference between the two instances is equal to k. In other words, J_2 is such that

$$\Pi(J_1 - J_2) = k.$$

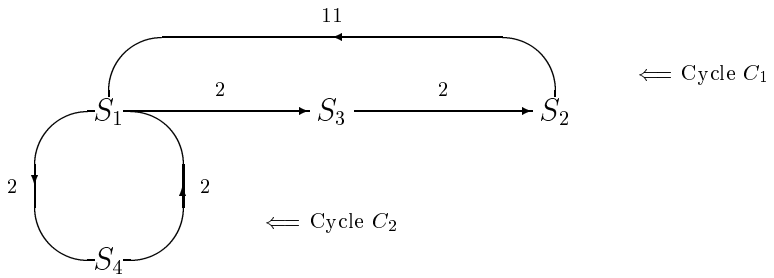

Figure 3: New weights obtained by $S_2^1 S_4^1$

Liu et al. [4] have a very nice way of determining J_2: let u be a vector such that $\Pi u = 1$ (remember that Π has relatively prime components). Then $J_2 = J_1 - ku$ satisfies $\Pi(J_1 - J_2) = k\Pi u = k$. Moreover, the vector u enables us to compute the new bounds for the loop indices. In the example let $u = (0, 1)$. Then the transformation S_2^1 amounts to shifting index j by one unit in statement S_2. Hence the second loop index j will loop from 1 to $N+1$ in statement S_2. But as it moves from 0 to N in statement S_1, the total range is extended from 0 to $N + 1$. Some iteration instances will be meaningless for some statements: in our example this is the case when $j = N + 1$ for statements S_1 and S_3 or when $j = 0$ for statements S_2 and S_4. Applying this process, we derive the new loop nest:

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

Suppose we were given this nest from the beginning: we would have computed the dependence matrix as

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

We check that $\Pi = (7, 1)$ is a scheduling vector with displacement $\text{disp}(\Pi) = 2$, hence we can achieve

$$\text{GSS}(\Pi) = \frac{7N + (N + 1)}{2} \leq 4N + 1,$$

nearly halving the execution time derived with the GSS approach.

We can now state more formally the ISM technique, as described by Liu et al [4]:

Index Shift Method: Given a scheduling vector Π with relatively prime components such that $\Pi D > 0$, let $Cycl$ denote the set of all cycles in the dependence graph where nodes are the statements S_1 to S_k and each edge (S_i, S_j) with dependence vector d_{ij} is

weighted by the value Πd_{ij}. For a cycle C in $Cycl$, let $T(C)$ denote its total weight and $k(C)$ denote its length.

The ISM technique enables us to replace the value of the reduction factor

$$\lambda_1 = \text{disp}(\Pi) = \min\{\Pi d_i, 1 \leq i \leq m\}$$

by the value

$$\lambda_2 = \min\{\lfloor T(C)/k(C) \rfloor, C \in Cycl\}.$$

Since the transformation is at the price of an increase in the range of the loop index that depends only on Π and not on the domain size, the improvement factor is very close to the ratio $\frac{\lambda_2}{\lambda_1}$.

4.2 An improvement of the Index Shift Method

Consider again Example 1. Remember that depending upon the shape of the domain, the GSS approach will determine either $\Pi_1 = (1, 0)$ or $\Pi_2 = (0, 1)$. Assume we want to use the ISM. For Π_1 we obtain the dependence graph of Figure 4(a), while similarly for Π_2 we get the graph of Figure 4(b).

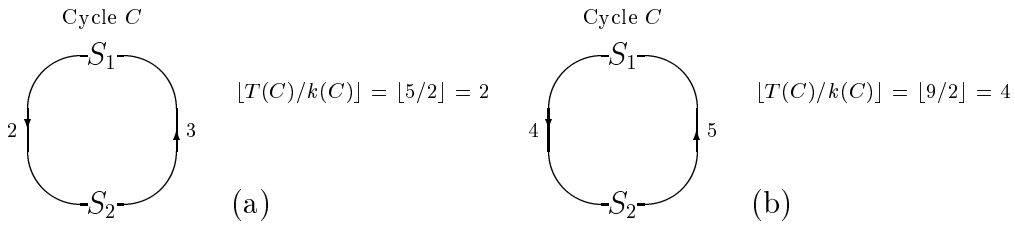

Figure 4: Dependence graph obtained by (a) Π_1 and (b) Π_2

We see that there is no possible improvement. However, using Π_1 say, we would like to replace $\text{disp}(\Pi_1) = 2$ by the real-valued average of the weights over the cycle, that is 2.5. The goal of this section is to show how we can deduce from Π_1 another scheduling vector Π_1^* that will achieve a reduction factor of 2.5.

In the general case, suppose we are given a cycle C of k statements S_1 to S_k with dependence matrix $D = (d_1, ..., d_k)$. Here vector d_j represents the dependence between statements S_j and $S_{j+1 \bmod k}$. Suppose we are also given a scheduling vector Π with relatively prime components and such that $\Pi D > 0$. Finally, suppose that the total weight of the cycle $T(C) = \Pi d_1 + ... + \Pi d_k$ is not divisible by k.

Of course, scaling Π by multiplying all its components by k would render $T(C)$ divisible by k, but would also violate the constraint of relatively prime components. So the idea is indeed to perform some scaling, but to use some perturbation afterwards to retrieve relatively prime components while satisfying the constraint $\Pi D > 0$.

Let r be a vector with relatively prime components and such that $\Pi r = 0$, and let s be a vector such that $rs = \pm 1$ with

$$sd_1 + ... + sd_k \geq 0.$$

Finding s from r is easy, and finding r from Π is done as follows. Compute the Hermite form of Π to get

$$\Pi = Q \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where Q is unimodular and let r be the second row of Q^{-1}: r has relatively prime components because Q^{-1} is unimodular, and $r\Pi = 0$ by construction, since

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = Q^{-1}\Pi.$$

Now, let $\Pi^* = \lambda k \Pi + s$. The weight $T^*(C)$ of cycle C with respect to Π^* is

$$T^*(C) = \lambda k T(C) + s(d_1 + .. + d_k) \geq \lambda k T(C)$$

hence $\lfloor T^*(C)/k \rfloor \geq \lambda T(C)$.

Now $r\Pi^* = \pm 1$, hence Π^* has relatively prime components. For λ large enough, $\Pi^* D > 0$, and after application of the ISM method we do achieve a reduction factor arbitrarily close to the real value of $T(C)/k$.

Rather than going on formally, let us apply the method to Example 1. Let us take $\Pi_1 = (1, 0)$. Then we have trivially

$$Q = \text{identity matrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and $r = s = (0, 1)$.

With $\Pi_1^* = 2\lambda\Pi_1 + s = (2\lambda, 1)$, we obtain the dependence graph of Figure 5 (a).

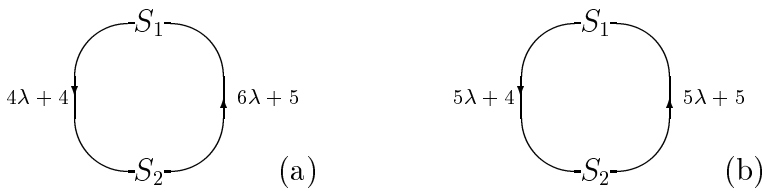

Figure 5: Dependence graph obtained (a) by Π_1^* and (b) after S_2^λ

The transformation S_2^λ produces the dependence graph of Figure 5 (b). Using $u = (0, 1)$, the second index j will loop from 0 to $N_2 + \lambda$. The resulting time will therefore be

$$\text{Time}(\Pi_1^*) = \frac{2\lambda N_1 + (N_2 + \lambda)}{5\lambda + 4} \leq \frac{N_1}{2.5} + 1$$

for large λ.

Similarly, if we take $\Pi_2 = (0, 1)$, we can obtain a Π_2^* that can achieve

$$\text{Time}(\Pi_2^*) \leq \frac{N_1}{4.5} + 1.$$

To conclude this section, we note that vector r has been introduced only to ease the presentation. In fact, computing Q^{-1} is needless: we just let s be the second column of Q (or its opposite).

5 Combining both methods

There is no reason a priori for the application of the ISM method or its refinement to the best GSS scheduling vector to give the best execution time. To see this, we return to Example 2.

5.1 GSS followed by ISM is not enough

We know that the best GSS scheduling vector is $\Pi = (7, 1)$ and for such a Π we had the dependence graph of Figure 2. Since for the cycle C_2 the total weight 4 is divisible by the length 2, we cannot improve on the original ISM method.

But why not try ISM with other scheduling vectors? Let $\Pi = (a, b)$ be an arbitrary scheduling vector. Hence $b \geq 1$ and $a \geq 6b + 1$, as seen earlier.

For cycle C_1 we get $T(C_1) = \Pi(d_1 + d_2 + d_3) = 2a + b$ and $k(C_1) = 3$. For cycle C_2 we get $T(C_2) = \Pi(d_4 + d_5) = a - 3b$ and $k(C_2) = 2$. For $b \geq 1$ and $a \geq 6b + 1$, we always have $(2a + b)/3 \geq (a - 3b)/2$, hence we want to minimize

$$\frac{(a + b)N}{\lfloor (a - 3b)/2 \rfloor}.$$

We can have this quantity as close to $2N$ as we want by letting $a = 2c + 1$ and $b = 1$ with c large (necessarily, $c \geq 3$). With $\Pi = (2c + 1, 1)$ we get the dependence graph of Figure 6.

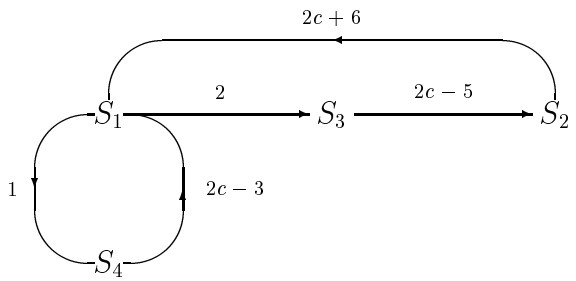

Figure 6: Dependence cycles obtained by $\Pi = (2c + 1, 1)$

By applying the transformation S_1^{2-c}, we get the dependence graph of Figure 7. Usage of $u = (0, 1)$ gives

$$\text{Time } \leq \frac{(2c+1)N + (N+2-c)}{c-1} \leq 2\frac{c+1}{c-1}N.$$

We conclude that time is close to $2N$, obtaining thus a factor 2 over GSS+ISM.

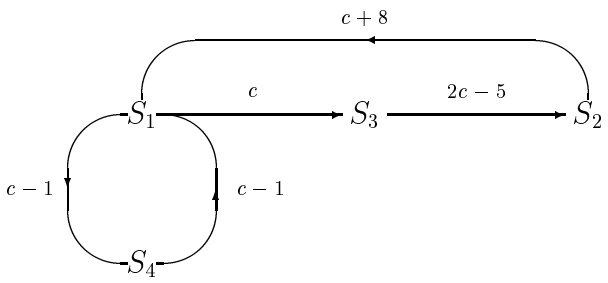

Figure 7: Dependence cycles obtained after S_1^{2-c}

5.2 A new optimization method

Consider a RUN with an index set Dom and a dependence matrix

$$D = \begin{pmatrix} d_1 & \dots & d_m \end{pmatrix}$$

where

$$d_i = \begin{pmatrix} d_{1i} \\ d_{2i} \\ \vdots \\ d_{ni} \end{pmatrix}$$

Let $Cycl$ denote the set of all cycles between the statements S_1 to S_k. For each cycle C in $Cycl$, let $k(C)$ denote its length.

The optimization problem can be stated as follows:

New Optimization Problem: Find a vector $\Pi = (\pi_1, \dots, \pi_n)$ such that

$$\begin{aligned} &(i) \quad \Pi D > 0 \\ &(ii) \quad \gcd(\pi_1, \dots, \pi_n) = 1 \end{aligned}$$

and which minimizes

$$\text{NEW}(\Pi) = \frac{\max\{\Pi I - \Pi J, I, J \in Dom\}}{\text{cycle}(\Pi)}$$

where

$$\text{cycle}(\Pi) = \min\{\lfloor \frac{T_\Pi(C)}{k(C)} \rfloor, C \in Cycl\}$$

with

$$T_{\Pi}(C) = \sum_{d \in C} \Pi d$$

Note that dependence vectors that are not involved in any cycle need not be taken into account for the computation of cycle(Π) (they still appear in condition (i)). This is clear from the ISM method, shifts can be freely applied to the corresponding edges in the dependence graph.

Note also that the problem is not scalable: due to the floor function, NEW(Π) $\neq$ NEW($\alpha\Pi$) for a nonzero constant α. However, our optimization problem is very close to the GSS problem. In [11], Shang and Fortes show how to solve GSS for a RUN or for a larger class of problems where the convex hull of the index set is a non-degenerate polyhedron. The idea is to partition the solution space into convex subcones, and to solve a linear fractional problem for each of these subcones. Two alternatives are proposed: either use a general linear programming approach or benefit from a less expensive method derived by the authors.

We can use the results of [11] as follows: we first remove condition (ii) that expresses that a scheduling vector must have relatively prime components. The problem can be expressed without floor functions and solved using Shang and Fortes' results. Finally, we perform some pertubation on the solution vector to retrieve condition (ii), as explained in section 4.2.

5.3 Quantifying the improvement factor

In this section, we show that our new method can outperform GSS followed by ISM by an arbitrary factor. Let λ be a positive integer, and consider the following example:

Example 3:

for $i = 0$ to N do
for $j = 0$ to N do
begin
{ Statement S_1 } $a(i,j) = f_1(b(i-1, j-\lambda), c(i-1, j+\lambda), \ldots)$
{ Statement S_2 } $b(i,j) = f_2(a(i, j-1), \ldots)$
{ Statement S_3 } $c(i,j) = f_3(a(i, j-1), \ldots)$
end

The dependences are the following:

$$S_1 \longrightarrow S_2 : d_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$S_2 \longrightarrow S_1 : d_2 = \begin{pmatrix} 1 \\ \lambda \end{pmatrix}$$

$$S_1 \longrightarrow S_3 : d_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$S_3 \longrightarrow S_1 : d_4 \;=\; \begin{pmatrix} 1 \\ -\lambda \end{pmatrix}$$

We obtain the following dependence matrix:

$$D = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & \lambda & 1 & -\lambda \end{pmatrix}.$$

The index domain is a square:

$$Dom = \{(i,j) \in Z^2, 0 \leq i,j \leq N\}.$$

There are two dependence cycles, namely $C_1 = (S_1, S_2)$ and $C_2 = (S_1, S_3)$.

Let us first compute the best scheduling vector with the GSS approach. We search for $\Pi = (a,b)$ such that

$$\text{(i) } \gcd(a,b) = 1$$
$$\text{(ii) } \Pi D > 0 \Longleftrightarrow \begin{cases} b \geq 1 \\ a \geq \lambda b + 1 \end{cases}$$

Note that a and b are necessarily positive. We get $\text{disp}(\Pi) = \min(b, a - \lambda b, a + \lambda b) = \min(a - \lambda b, b)$ and we want to minimize the quantity

$$\text{GSS}(\Pi) = \frac{(a+b)N}{\text{disp}(\Pi)}.$$

The following straightforward case analysis shows that $\Pi = (\lambda + 1, 1)$, for which $\text{GSS}(\Pi) = (\lambda + 2)N$, is the best solution:

i. if $b = 1$, then minimize $\text{GSS}(\Pi) = (a+1)N$, hence take $a = \lambda + 1$

ii. if $b \geq 2$, then consider two subcases:

   (a) if $\lambda b + 1 \leq a \leq (\lambda + 1)b - 1$, then $\text{disp}(\Pi) = a - \lambda b$, and $\text{GSS}(\Pi) = \frac{a+b}{a-\lambda b}N$, hence take $a = (\lambda+1)b - 1$ and get $\text{GSS}(\Pi) = \frac{(\lambda+2)b-1}{b-1}N \geq (\lambda + 2)N$

   (b) if $(\lambda + 1)b + 1 \leq a$, then $\text{disp}(\Pi) = b$, and $\text{GSS}(\Pi) = \frac{a+b}{b}N$, hence take $a = (\lambda+1)b + 1$ and get $\text{GSS}(\Pi) = \frac{(\lambda+2)b+1}{b}N \geq (\lambda + 2)N$. (Note that $a = (\lambda+1)b$ is excluded because $\gcd((\lambda+1)b, b) = b \geq 2$.)

Next, draw a directed graph (Figure 8) whose nodes are the statements S_1 to S_3 and whose edges are weighted by the values $e_i = \Pi d_i$.

Because of cycle C_2, there is no possible improvement using ISM. However, if we solve the new optimization problem, we have to search for $\Pi = (a,b)$ such that (i) and (ii) hold (note that a and b are necessarily positive). We get $\text{cycle}(\Pi) = \min(\lfloor \frac{b+(a+\lambda b)}{2} \rfloor, \lfloor \frac{b+(a-\lambda b)}{2} \rfloor)$ $= \lfloor \frac{a+(1-\lambda)b}{2} \rfloor$, and we want to minimize the quantity

$$\text{NEW}(\Pi) = \frac{(a+b)N}{\text{cycle}(\Pi)}.$$

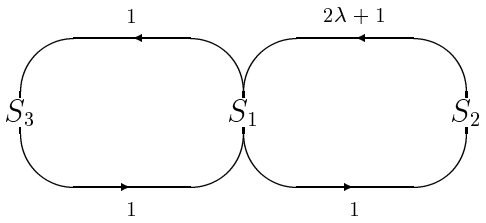

Figure 8: Dependence cycles of Example 3 with GSS

We let $\lambda = 2\gamma - 1$ and we take $\Pi = (a, b) = (4\gamma, 1)$ as a scheduling vector (we check that $a \geq \lambda b + 1$, i.e., $4\gamma \geq (2\lambda - 1).1 + 1$). We obtain the new dependence graph shown in Figure 9.

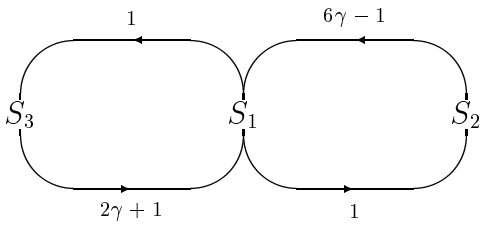

Figure 9: Dependence cycles with $\Pi = (4\gamma, 1)$

We have cycle$(\Pi) = \gamma + 1$. Next we perform the transformation S_2^{γ} S_3^{γ} (see Figure 10).

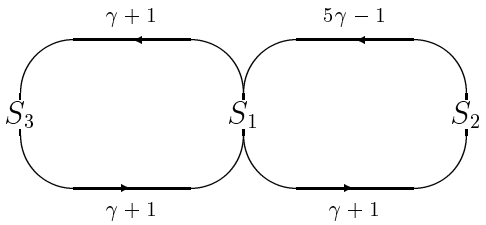

Figure 10: Dependence cycles obtained after S_2^{γ} S_3^{γ}

We use vector $u = (0, 1)$ such that $\Pi u = 1$ to compute the new index bounds:

$$0 \leq i \leq N$$
$$\begin{cases} 0 \leq j \leq N & \text{(statement } S_1\text{)} \\ \gamma \leq j \leq \gamma + N & \text{(statements } S_2 \text{ and } S_3\text{)} \end{cases}$$

The new domain is then

$$Dom = \{(i, j) \in Z^2, 0 \leq i \leq N, 0 \leq j \leq N + \gamma\}.$$

We get

$$\text{NEW}(\Pi) = \frac{4\gamma N + (N+\gamma)}{\gamma+1} \leq 4N+1.$$

Hence we gain a factor close to $\frac{\lambda+2}{4} = O(\lambda)$ over GSS+ISM, and this factor can be arbitrary high, as was claimed.

6 Conclusion

Nested loop structures offer the most fruitful sources of parallelism in serial programs, and it is therefore of paramount importance that the analysis necessary for such programs be both precise and efficient, by exploiting the potential of parallel execution of loop statements to its full extent.

Several loop transformations techniques have been designed to extract parallelism from nested loop structures. We have reviewed two important approaches, known as *Generalized Cycle Shrinking* presented by Shang, O'Keefe and Fortes and *the Index Shift Method* introduced by Liu, Ho and Sheu. We have used an illustrative example (Example 2) to show the gains that can be obtained. The quadratic sequential time is reduced to $8N$ through GSS and to $4N$ through GSS followed by ISM. One result of the paper is an improvement of the index shift method, for a given scheduling vector Π. The main result of the paper is a new methodology that permits to combine cycle shrinking techniques with the index shift method. Through this combination, the illustrative example renders a further factor of 2, giving a time of $2N$. The combination of the two techniques gave rise to a new optimization method that produces the best scheduling vector, and we have shown that we can outperform previous results by an arbitrary speedup factor.

Acknowledgments

We would like to thank Alain Darte for his comments on a first version of the paper.

References

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

[2] M.L. Dowling, "Optimal code parallelization using unimodular transformations", *Parallel Computing*, 16, 1990, pp. 157 - 171.

[3] C.E. Leiserson and J.B. Saxe, "Optimizing synchronous systems", *Journal of VLSI Computer Systems*, Vol. 1, April, 1983, pp. 41 - 67.

[4] L.S. Liu, C.W. Ho, and J.P. Sheu, "On the parallelism of nested for-loops using index shift method", Proceedings of International Conference on Parallel Processing, August 1990, pp. II-119 - II-123.

[5] J.K. Peir and R. Cytron, "Minimum distance: a method for partitioning recurrences for multiprocessors", *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989, pp. 1203 - 1211.

[6] C.D. Polychronopoulos, "Compiler optimization for enhancing parallelism and their impact on architecture design", *IEEE Transactions on Computers*, Vol. 37, No. 8, August 1988, pp. 991 - 1004.

[7] C.D. Polychronopoulos, *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston, 1988.

[8] P. Quinton and Y. Robert, *Algorithmes et Architectures Systoliques*, Masson, Paris, 1989.

[9] W. Shang and J.A.B. Fortes, "Time optimal linear schedules for algorithms with uniform dependences", *Proceedings of International Conference on Systolic Arrays*, May 1988, pp. 393 - 402.

[10] W. Shang and J.A.B. Fortes, "Time optimal and conflict-free mappings for uniform dependence algorithms into lower dimensional processor arrays", Proceedings of International Conference on Parallel Processing, August 1990, pp. I-101 - I-110.

[11] W. Shang and J.A.B. Fortes, "Time optimal linear schedules for algorithms with uniform dependencies", *IEEE Transactions on Computers*, Vol. 40, No. 6, June 1991, pp. 723-742.

[12] W. Shang, M.T. O'Keefe, J.A.B. Fortes, "On loop transformations for generalized cycle shrinking", Proceedings of International Conference on Parallel Processing, August 1991, pp. II-132 - II-141.

[13] W. Shang, M.T. O'Keefe, J.A.B. Fortes, "Generalized cycle shrinking", in: *Parallel Algorithms and VLSI Architectures II*, P. Quinton and Y. Robert (editors), North Holland, 1991.

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

[15] M. Wolfe, "Data dependence and program restructuring", *The Journal of Supercomputing 4*, 1990, pp. 321 - 344.