RPUSM: An Effective Instruction Scheduling Method for Nested Loops

Yi-Hsuan Lee, Ming-Lung Tsai and Cheng Chen
Department of Computer Science and Information Engineering
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Tel: (8863) 5712121 EXT: 54734, Fax: (8863) 5724176
E-mail: {yslee, mltsai, cchen}@csie.nctu.edu.tw

Abstract

Multi-dimensional systems are widely used to model scientific applications such as image processing, geophysical signal processing and fluid dynamics. Such systems usually contain repetitive groups of operations represented by nested loops. The optimization of such loops, considering processing resource constraints, is required in order to improve their computational time. Push_Up Scheduling Method (PUSM) is an effective technique that can get the shortest schedule table and fully utilize resources. It contains two main parts: schedule table construction and retiming base selection. In our analysis, PUSM can obtain optimal solution in the former part, but lacks a complete selecting process and uses more constrained conditions in the later part. Hence, in this paper, we propose a Relax Push_Up Scheduling Method (RPUSM) to overcome its shortcomings. In RPUSM, we inherit the former part from PUSM, and replace the later part by a complete and more relax conditions to select more appropriate retiming base. According to our analysis, RPUSM can not only get better performance than PUSM in some cases, but also preserve the advantages of PUSM, such as shortest schedule table, fully resource utilization, and polynomial scheduling time.

Keywords: *Instruction scheduling, Push-up Scheduling, Retiming*

1. Introduction

Multi-dimensional systems are widely used to model scientific applications such as image processing, geophysical signal processing and fluid dynamics [3, 5, 7]. These systems usually contain repetitive groups of operations represented by nested loops. Since nested loops are the time-critical sections in such computation-intensive applications, their execution time usually dominates the entire computational performance. To optimize the execution rate of such applications, we need to explore the embedded parallelism in repetitive patterns of a loop [13, 15].

Traditional scheduling methods can be divided into five categories [10]: (1) Integer Linear Programming (ILP) [2, 4], (2) List Scheduling technique [9], (3) Probability Based technique [19], (4) Randomized Searching Based technique [12], and (5) Transformation Based technique [1, 8, 11]. It usually uses Multi-dimensional Data-Flow Graph (MDFG) to represent iteration, and some techniques such as retiming and unfolding are used to regroup operations in iterations [1, 8, 11]. After applying

these scheduling methods, we can get a new iteration structure with higher parallelism embedded. Among them, *Push_Up Scheduling Method* (PUSM) proposed by [17] is an effective technique. It not only can get the shortest schedule table and fully utilize resources, but also runs in polynomial time. However, it still has some shortcomings according to our analysis. Therefore, we propose a *Relax Push_Up Scheduling Method* (RPUSM) to overcome its shortcomings in this paper.

In the scheduling algorithm of PUSM, we can find it contains two main parts: schedule table construction and retiming base selection. Since PUSM can get the shortest schedule table and fully utilize resources, it already achieves the optimal result in the former part. In the later part, however, it lacks a complete selecting process and its selecting conditions are more constrained. In our method, hence, we inherit the former part from PUSM, and propose a complete and more relax conditions to select more appropriate retiming base. According to our analysis, RPUSM can not only get better performance than PUSM in some cases, but also preserve the original advantages of PUSM, such as shortest schedule table, fully resource utilization, and polynomial scheduling time.

The remaining of this paper is organized as follows. Section 2 introduces some fundamental backgrounds and related work. The design issues and principles of our Relax Push_Up Scheduling Method are introduced in Section 3. In Section 4, we give some preliminary analysis of our method to demonstrate its figure of merits. Finally, some concluding remarks are given in Section 5.

2. Fundamental Background and Related Work

Traditional instruction scheduling method can be divided into five categories [10]. PUSM belongs to Transformation Based technique, which is the most popular one. Scheduling methods belonged to this category restructure the loop body to explore the instruction level parallelism, and using the modified repetitive pattern can decrease the overall execution time [1, 8, 11]. Most of them not only can obtain good results, but also need less time and space complexity. In this section, we will briefly survey basic principles and scheduling algorithm of PUSM.

2.1 Basic Principles [13, 15, 17]

Most transformation based scheduling methods model the nested loop as *Multi-dimensional Data-Flow Graph* (MDFG), which is defined as follows.

Definition 2.1 An MDFG $G = (V, E, d, t)$ is a node-weighted and edge-weighted directed graph, where V is the set of computation nodes, E is the set of dependence edges, d is a function from E to Z^n, representing the multi-dimensional delays between two nodes, where n is the number of dimensions, and t is a function from V to the positive integers, representing the computation time of each node.

Figure 1 is an example of a nested loop and its corresponding MDFG. An equivalent *cell dependence graph* (cell DG) of an MDFG is the directed acyclic graph showing the dependencies between copies of nodes representing the MDFG. The cell dependence graph of the MDFG in Figure 1(b) is shown in Figure 2(a).

An MDFG $G = (V, E, d, t)$ is *realizable* if there exists a schedule vector s such that $s \cdot d \geq 0$,

```
for i = 1 to m begin
 for j = 1 to n begin
   D [i, j] = B [i–1, j] × C [i–1, j+2] ;
   A [i, j] = D [i, j] × 0.5 ;
   B [i, j] = A [i, j] + 1 ;
   C [i, j] = A [i, j–1] + 2 ;
 end
end
```

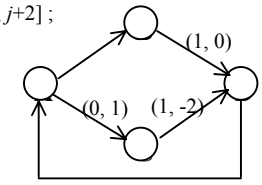
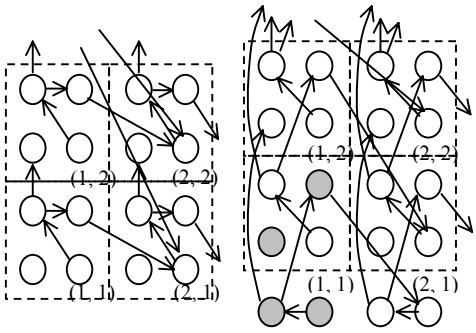


Figure 1. Nested loop and MDFG.



(a) (b)

Figure 2. (a) Cell DG, (b) retimed cell DG.

where d are loop-carried dependences in G. A *schedule vector s* is the normal vector for a set of parallel equitemporal hyperplanes that define a sequence of execution [6]. An *iteration* is equivalent to the execution of each node in V exactly once. The period during which all computation nodes in an iteration are executed, according to existing data dependences and without resource constraints, is called a *cycle period*. The cycle period of an MDFG is the maximum computational time among paths that have no delay. It can be shown that the cycle period dominates the entire execution time of a nested loop. Hence, the goal of most scheduling methods is to try to minimize the cycle period.

Retiming is a popular technique to reduce the cycle period of an MDFG [8]. A *multi-dimensional retiming r* is a function from V to Z^n that redistributes the nodes in the cell DG created by the replication of an MDFG $G = (V, E, d, t)$. A new MDFG $G_r = (V, E, d_r, t)$ is created after applying retiming function r, such that each iteration still has one execution of each node in G. The retiming vector $r(u)$ of a node u represents the offset between the original iteration and the one after retiming. The delay vectors change accordingly to preserve dependencies. The definitions and properties of retiming are shown in the following definition.

Definition 2.2 Given any MDFG $G = (V, E, d, t)$, retiming function r, and retimed MDFG $G_r = (V, E, d_r, t)$, we define the retimed delay vector for every edge, path, and cycle respectively by the following formulas:

(a) $d_r(e) = d(e) + r(u) - r(v)$ for every edge $u \xrightarrow{e} v$, $u, v \in V$ and $e \in E$.

(b) $d_r(p) = d(p) + r(u) - r(v)$ for every path $u \xrightarrow{p} v$, $u, v \in V$ and $p \in G$.

(c) $d_r(l) = d(l)$ for any cycle $l \in G$.

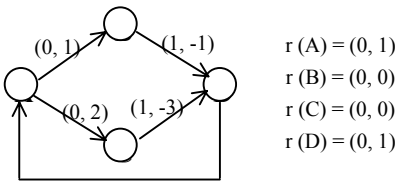


Figure 3. Retimed MDFG.

Figure 3 and 2(b) show the retimed MDFG and cell DG, where the nodes originally belonging to iteration (1, 1) are marked. A *prologue* is the set of instructions that are moved on each dimension and must be executed to provide the necessary data for the iterative process. An *epilogue* is the other extreme of the cell DG, were a complementary set of instructions will be executed to complete the process. Considering that the entire problem consists of a large number of iterations, the time required to run the prologue and epilogue are negligible [17].

Although retimed MDFG contains smaller

cycle period than original one, we must insure that the retimed MDFG is still realizable. A *legal multi-dimensional retiming r* is defined below.

Definition 2.3 Given a realizable MDFG G, a legal multi-dimensional retiming for G is the multi-dimensional retiming function r that transforms G in G_r, such that G_r is still realizable.

As we mention before, an MDFG G is realizable if there exists a schedule vector s such that $s \cdot d \geq 0$, where d are loop-carried dependences in G. Similarly, a schedule vector s can be found if the retimed MDFG G_r is realizable, too. The following theorem describes the relationship between schedule vector s and legal multi-dimensional retiming function r.

Theorem 2.1 Let $G = (V, E, d, t)$ be a realizable MDFG, s a schedule vector such that $s \cdot d \geq 0$ where d is non-zero dependence in G, and $u \in V$ a node with all the incoming edges having non-zero delays. A legal multi-dimensional retiming function $r(u)$ is any vector orthogonal to s.

Proof: It can be found in [15]. □

Corollary 2.2 Let $G = (V, E, d, t)$ be a realizable MDFG, s a schedule vector that realizes G, and $u \in V$. If $r(u)$ is a legal multi-dimensional retiming function, then $(k \times r)(u)$, where k is a positive integer, is also a legal multi-dimensional retiming.

Proof: It can be found in [15, 17].□

According to Theorem 2.1, we can find a legal multi-dimensional retiming function from the original MDFG directly. We have introduced the basic principles using in PUSM above, and the detail description of PUSM will be contained in the next subsection.

2.2 Push_Up Scheduling Method [17]

The main principle of PUSM is to schedule operations as early as possible, so that it obtains a schedule with minimum length under some particular resource constraints. At first, we define a *schedulable node* below:

Definition 2.4 (Scheduling Conditions) Given an MDFG $G = (V, E, d, t)$ and a node $u \in V$, u is *schedulable* at a control step cs if it satisfies one of the conditions below:

1. u has no incoming edges;
2. all incoming edges of u have a non-zero multi-dimensional delay;
3. all the predecessors of u, connected to u by a zero-delay edge, have been scheduled to earlier control steps.

Retiming technique will change the delay value of an edge, so it can be used to change a zero-delay edge to a non-zero one and vice versa. Therefore, if a node has some incoming edges with zero-delay and is not schedulable, we can use retiming technique to make it become schedulable. PUSM uses this feature to schedule each node as early as possible to fully utilize the resources, so it can always get a shortest schedule table under some particular resource constraints.

The first step of PUSM is to select a suitable *retiming base r*. The selecting conditions is similar to Theorem 2.1, but it uses $s \cdot d > 0$ and $r \perp s$ to guarantee that the retimed MDFG is strictly realizable. After applying PUSM, we will get a schedule table with minimum length and the retiming degree of each node. The retiming function of each node is the production of the retiming base r and its retiming degree. The maximum retiming degree is called *retiming depth d*. The complete PUSM algorithm can be found in [17].

Although PUSM can always obtain a shortest schedule table, it still has two shortcomings. The first one is that it doesn't contain a complete process to select the retiming base, and the second one is that its selecting condition of retiming base is more constrained. Hence, we propose a *Relax Push_Up Scheduling Method* (RPUSM) to overcome these two shortcomings in the next section.

3. Relax Push-Up Scheduling Method

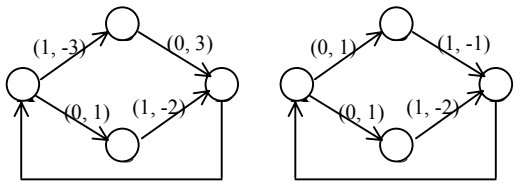
In the scheduling algorithm of PUSM, we can find it contains two main parts. One is how to obtain the schedule table and retiming degree of each node, and the other is how to select the retiming base. In the following, we give an example to illustrate its constrained selecting condition.

Given a nested loop with depth two and the corresponded MDFG as shown in Figure 1, and here we assume variables *m* and *n* equal to 10 and 8 respectively. After applying PUSM, we will obtain retiming base (1, -3) and retiming degree of each node. If both multiply and add operations need 1 cycle to execute, we can calculate that the overall execution time is 195 cycles. Noted that the execution sequence is not regular. On the other hand, if we violate the selecting condition in PUSM and select retiming base (0, 1), the overall execution time will decrease to 170 cycles. The schedule table and two retimed MDFGs are shown in Figure 4. From this example, we can see that the retiming base selected by PUSM is not always optimal. Therefore, our RPUSM will focus on the selecting algorithm design of retiming base. Since the nested loops used in scientific applications are usually two-dimensional, we use

CS	Mul.	Adder
P	D	C
P	A	
1	D	C
2	A	B
E		B

P : prologue E : epilogue

(a)



r (A) = (1, -3) r (B) = (0, 0) r (C) = (1, -3) r (D) = (1, -3)

(b)

r (A) = (0, 1) r (B) = (0, 0) r (C) = (0, 1) r (D) = (0, 1)

(c)

Figure 4. (a) Schedule table, (b) retimed MDFG, (c) retimed MDFG.

nested loop with depth two as an example to explain RPUSM clearly. Nevertheless, RPUSM can be extended to cover nested loop with depth more than two easily similar to PUSM.

Before introducing RPUSM, we use Lemma 3.1 and 3.2 to present the influence of schedule vector on execution time. Additional variables used in these Lemmas are defined at first. $Length_{list}$ and $length_{PUSM}$ are the schedule lengths of loop body produced by *List Scheduling Method* [9] or PUSM. List Scheduling Method is a simple scheduling method without restructuring the loop body, so it usually can't obtain the shortest schedule length. *Prologue* and *epilogue* are the time needed to execute the extra codes produced by PUSM for the same name, and *d* is the corresponding retiming depth.

Lemma 3.1 Given a nested loop with depth two, and its loop bounds of outer and inner loops are *m* and *n* respectively. We use PUSM to schedule it on uniprocessor architecture. If the schedule vector used in PUSM is (1, 0), then the entire

execution time is $length_{PUSM} \times m(n - d) + (prologue + epilogue) \times m$.

Proof: The schedule vector s is (1, 0), that corresponds to the normal execution sequence, so we can simply select retiming base (0, 1) that is orthogonal to s. Because the retiming base and retiming depth are (0, 1) and d, md iterations should be moved into prologue and epilogue. After applying PUSM, it produces $m(n - d)$ restructured loop bodies and m pairs of prologue and epilogue. Since the system architecture is uniprocessor, it is directly that the execution time is $length_{PUSM} \times m(n - d) + (prologue + epilogue) \times m$. □

Lemma 3.2 Given a nested loop with the same assumption as Lemma 3.1, and use PUSM to schedule it on uniprocessor architecture. If the schedule vector used in PUSM is (s_1, s_2), both s_1 and s_2 are positive integers, then the entire execution time is $length_{PUSM} \times (m - s_2d)(n - s_1d) + (prologue + epilogue) \times (s_1m + s_2n - s_1s_2 - 2ds_1s_2) + length_{list} \times s_1s_2d(d + 1)$.

Proof: The schedule vector s is (s_1, s_2), both s_1 and s_2 are positive integers, so we can simply select retiming base $(s_2, -s_1)$ that is orthogonal to s. Because the execution sequence is not normal, the scheduling result is much complex. Figure 5(b) shows the changed iteration space after applying PUSM with retiming depth d and retiming base $(s_2, -s_1)$, and the thick lines represent the equitemporal hyperplanes. In this figure, we can see the iteration space is partitioned into three regions. Region A contains $(m - s_2d)(n - s_1d)$ iterations, which are loop bodies produced by PUSM. Region B contains $d(s_1m + s_2n - s_1s_2 - 2ds_1s_2)$ iterations, which forms m pairs of prologue and epilogue. Region C contains the remainder $s_1s_2d(d + 1)$ iterations, which must execute using List Scheduling Method because it is out of the nested loop. Therefore, the entire execution time is $length_{PUSM} \times (m - s_2d)(n - s_1d) + (prologue + epilogue) \times (s_1m + s_2n - s_1s_2 - 2ds_1s_2) + length_{list} \times s_1s_2d(d + 1)$. □

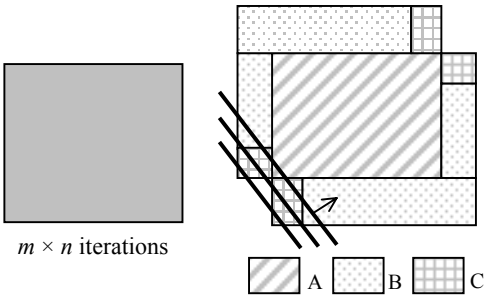


Figure 5. (a) Original, (b) modified iteration space.

According to above two Lemmas, Theorem 3.1 shows that using schedule vector (1, 0) always can get shorter execution time.

Theorem 3.1 Given a nested loop with the same assumption as Lemma 3.1, and use PUSM to schedule it with schedule vector (1, 0) and (s_1, s_2) respectively. The overall execution time with schedule vector (1, 0) is always shorter than that of it with schedule vector (s_1, s_2).

Proof: Execution time with these two schedule vectors can be obtained from Lemma 3.1 and 3.2 directly. For the convenience, we use variables T_1 and T_2 to represent them respectively. We have the following expressions:

$T_1 = length_{PUSM} \times m(n - d) + (prologue + epilogue) \times m$

$T_2 = length_{PUSM} \times (m - s_2d)(n - s_1d) + (prologue + epilogue) \times (s_1m + s_2n - s_1s_2 - 2ds_1s_2) + length_{list} \times s_1s_2d(d + 1)$

$T_2 - T_1 = length_{PUSM} \times d(m + s_1s_2d - s_1m - s_2n) + (prologue + epilogue) \times (s_1m + s_2n - s_1s_2 - 2ds_1s_2 - m) + length_{list} \times d(s_1s_2d + s_1s_2)$ (1)

Since PUSM can always obtain the shortest schedule table but List Scheduling Method can't, we have $length_{list} \geq length_{PUSM}$. Because $length_{list} \geq length_{PUSM}$, formula (1) can be rewritten as

$$T_2 - T_1 \geq length_{PUSM} \times d(m + 2s_1s_2d - s_1m - s_2n + s_1s_2) + (prologue + epilogue) \times (s_1m + s_2n - s_1s_2 - 2ds_1s_2 - m) \quad (2)$$

Since prologue and epilogue contain d iterations and PUSM can get the shortest schedule table, we have $prologue + epilogue \geq length_{PUSM} \times d$. Because $prologue + epilogue \geq length_{PUSM} \times d$, formula (2) can be rewritten as

$$T_2 - T_1 \geq length_{PUSM} \times d(m + 2s_1s_2d - s_1m - s_2n + s_1s_2) + length_{PUSM} \times d(s_1m + s_2n - s_1s_2 - 2ds_1s_2 - m) = 0$$

Hence, we have $T_2 \geq T_1$ and complete this proof. □

From Theorem 3.1, we should select schedule vector (1, 0) as far as possible when we use PUSM to schedule a nested loop with depth two. It can not only decrease overall execution time, but also simplify the complexity of the retimed nested loop. In the following, we will analyze the conditions that can select schedule vector (1, 0).

Given a two-dimensional MDFG $G = (V, E, d, t)$ and use PUSM to schedule it. If we want to select schedule vector (1, 0), we have to guarantee that the retimed MDFG $G_r = (V, E, d_r, t)$ with retiming base (0, 1) is still realizable. As mentioned in Section 2.1, an MDFG $G = (V, E, d, t)$ is realizable if there exists a schedule vector s such that $s \cdot d \geq 0$, where d are loop-carried dependences in G. In the other words, an MDFG is realizable if there exists an execution sequence that makes all dependences be flow-dependences. That is to say that we have to guarantee the retimed MDFG G_r with retiming base (0, 1) doesn't contain any anti-dependence.

Since we use retiming base (0, 1), the retiming function of each node will be the model of $(0, a)$ where $a \geq 0$. According to the definition of retiming technique, only dependence with distance $(0, a)$, $a \geq 0$, may become anti-dependence after retiming. Therefore, we only have to check dependences with this model in the original MDFG. Theorem 3.2 shows the conditions that we can select schedule vector (1, 0) and retiming base (0, 1) respectively.

Theorem 3.2 Given a two-dimensional MDFG $G = (V, E, d, t)$ and use PUSM to schedule it. Without loss of generality, we consider G only containing flow-dependences. After applying PUSM, we will obtain the retiming degree $rd(v)$ of each node $v \in V$. We can select schedule vector s equals to (1, 0) under the following conditions:

1. G doesn't contain any dependence with distance $(0, a)$, $a > 0$.
2. G contains dependence $u \xrightarrow{(0,a)} v$ and $rd(u) + a \geq rd(v)$, where $u, v \in V$, $a > 0$.

Proof: Since we only consider flow-dependence, the dependence distances in G can be classified into four models: $(0, a)$, $(b, 0)$, (c, d) and $(e, -f)$, where the six variables are all positive. Our goal is to shown that the retimed MDFG $G_r = (V, E, d_r, t)$ with retiming base (0, 1) is still realizable with schedule vector (1, 0).

In the first condition, G doesn't contain dependence with distance $(0, a)$, $a > 0$. It is to say that dependence distances in G can be classified into three models: $(b, 0)$, (c, d) and $(e, -f)$, where the five variables are all positive. Because the retiming function of each node will be the model of $(0, a)$, $a \geq 0$, dependence

distances in G_r can also be classified into three models as above. Since the inner products of (1, 0) and these three models are all positive, G_r is realizable with schedule vector (1, 0).

In the second condition, G may contain four dependence distance models: (0, a), (b, 0), (c, d) and (e, $-f$), where the six variables are all positive. The last three models are the same as the first condition, so we only consider dependence with distance (0, a). According to the definitions of this condition and retiming technique, dependence (0, a) becomes to (0, a + rd (u) – rd (v)) after applying PUSM. Because rd (u) + $a \geq rd$ (v), the inner product of (1, 0) and (0, a + rd (u) – rd (v)) is still positive.

Therefore, we can select schedule vector (1, 0) and retiming base (0, 1) if G satisfies these two conditions. □

As mentioned above, PUSM contains two main parts: schedule table construction and retiming base selection. The former part is already achieved to the optimal solution, but the later part still can be improved. Hence, in our *Relax Push_Up Scheduling Method* (RPUSM), we replace the later part by the result of Theorem 3.2 and inherit the former part to preserve its original advantages. The complete algorithm of RPUSM is shown in Figure 6.

```
1  input : MDFG G = (V, E, d, t)
2  output : MDFG Gr = (V, E, dr, t), schedule table S
3  ES(∀u ∈ V) ← 0; MC(∀u ∈ V) ← 0;
4  MCmax ← 0; QueueV ← φ;
5  ∀e ∈ E, E ← E − {e, s.t. d(e) ≠ (0,0)};
6  QueueV ← QueueV ∪ {u ∈ V, s.t. InDEGREE(u) = 0};
7  while QueueV ≠ φ
8    GET(u, QueueV);
9    if AVAIL(fu) < ES(u)
10     MC(u) ← MC(u) + 1;
11     MCmax ← max{MC(u), MCmax};
12   endif
13 ES(u) ← AVAIL(fu);
14 ASSIGN u to fu at control step ES(u);
15 ∀v such that u → v
16     INDEGREE(v) ← INDEGREE(v) − 1;
17     ES(v) ← max{ES(v), ES(u) + t(u)};
18     MC(v) ← max{MC(v), MC(u)};
19     if INDEGREE(v) = 0
20       QueueV ← QueueV ∪ {v};
21    endif
22 endwhile
23 ∀u ∈ V, rd(u) ← MCmax − MC(u);
24 if !∃d(e) = (0, a)  for a > 0
25    s = (1,0); r = (0,1);
26 elseif ∃u --(0,a)--> v and rd(u) + a ≥ rd(v)
27    s = (1,0); r = (0,1);
28 else select s = (s1, s2) and r ⊥ s;
29 endif
30 ∀u ∈ V, r(u) ← rd(u) × r;
31 return Gr, S;
```

Figure 6. Complete scheduling algorithm of RPUSM.

4. Preliminary Performance Analysis

In this section, we use six examples to present our method. These examples are selected from [14, 16, 18, 20], which represent DSP applications. Figure 7 shows their MDFGs, where we use rectangle and circle to represent multiplication and addition respectively. According to the number of multiplications and additions in each application, we use different number of multipliers and adders to balance their utilization. Assume that both multiplication and addition require one clock cycle to complete and every application contains 30 × 30 iterations. The resources constraints and scheduling results are shown in Table 1 and 2.

In Table 2, we can find that RPUSM can't benefit from examples Transmission Lines and IIR Filter because schedule vectors selected in PUSM and RPUSM are the same. According to selecting conditions of PUSM and RPUSM, if original MDFG doesn't contain dependence with distance (0, a), a > 0, they will select the same schedule vector (1, 0). RPUSM can't obtain

Table 2. Scheduling result.

	PUSM		RPUSM	
	Schedule vector	Execution time	Schedule vector	Execution time
Filter 1 [14]	(1, 1)	1916	(1, 0)	1802
Filter 2 [16]	(1, 1)	2759	(1, 0)	2701
Model A [20]	(3, 1)	4143	(1, 0)	3605
Transmission Lines [17]	(1, 0)	3603	(1, 0)	3603
IIR Section [18]	(1, 1)	3773	(1, 0)	3603
IIR Filter [17]	(1, 1)	3881	(1, 1)	3881

better performance in this situation. Otherwise, since PUSM won't select schedule vector (1, 0) but RPUSM will, RPUSM may get shorter execution time.

In the following, we use example Model A to compare the performance between PUSM and RPUSM with the scale of application size. Figure 8 shows the growing up of execution time and the percentage of their difference. In this figure, we can find that only if RPUSM can benefit from PUSM, its execution time will be always shorter because it uses less time to execution prologue and epilogue. But repetitive patterns will dominate the entire execution time with the growing of application size, so its percentage of difference is decreasing gradually.

5. Concluding Remarks

In this paper, we have proposed a Relax Push_Up Scheduling Method to schedule a nested loop on uniprocessor architecture, and compare its performance with original Push_Up Scheduling Method. RPUSM contains two main parts like PUSM: schedule table construction and retiming base selection. The former part is inherited from PUSM, since it already can achieve optimal solution. The later one contains a complete process and a more relax condition to select retiming base, which is the shortcomings of original PUSM. Hence, RPUSM can not only preserve the advantages of PUSM such as shortest schedule length and fully resource utilization, but also get better performance in some cases.

In addition to previous features, there are still several promising issues in future researches. The first one is the scheduling algorithm modification. According to our observation, although PUSM can always get shortest schedule table, it usually produces much longer prologue and epilogue compared with *Multi-dimensional Rotation Algorithm* [17]. This feature may indirectly affect the overall execution time, especially when the schedule vector is not normal. Therefore, we can try to modify its scheduling algorithm to get a new schedule table, which with shortest length and smaller prologue and epilogue. The second one is the extension to multiprocessor architecture. Intuitively, PUSM and RPUSM can be used in multiprocessor architecture directly, which replaces the resource constraint from function unit to processor. This is a simple extension method, but it may cause slight communication overhead and loss the advantage of locality. Hence, we can try to combine PUSM (RPUSM) with loop transformation techniques such as loop skewing and permutation to form a new *Two-Level Scheduling Method*. Since PUSM (RPUSM) and loop transformation techniques are independent of the expected granularity of parallelism, combining these two techniques will be a very instructing research topic in the future.

Acknowledgments

This research was supported by the National Science Council of the Republic of China under contract numbers: NSC 89-2213-E009-200.

Reference

[1] L. F. Chao and E. H. M. Sha, "Static Scheduling of Uniform Nested Loops", *Proc. of 7th International Parallel Processing Symposium*, pp. 1421-1424, Apr. 1993.

[2] C. H. Gebotys, "Optimal Synthesis of Multichip Architectures", *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 238-241, Sep. 1992.

[3] Y. C. Hsu and Y. L. Jeang, "Pipeline Scheduling Techniques in High-Level Synthesis", *Proc. of 6th Annual IEEE International ASIC Conference and Exhibit*, pp. 396-403, Sep. 1993.

[4] C. T. Hwang, J. H. Lee and Y. C. Hsu, "A Formal Approach to the Scheduling Problem in High-Level Synthesis", *IEEE Trans. on Computer-Aided Design*, Vol. 10, Issue 4, pp. 464-475, April 1991.

[5] S. Y. Kung, **VLSI Array Processors**, Prentice Hall, 1988.

[6] L. Lamport, "The Parallel Execution of DO Loops", *Comm. ACM SIGPLAN*, Vol. 17, No. 2, pp. 82-93, Feb. 1974.

[7] T. F. Lee, Allen C. H. Wu, Daniel D. Gajski and Y. L. Lin, "An Effective Methodology for Functional Pipelining", *Proc. of IEEE International Conference on Computer-Aided Design*, pp. 203-233 Nov. 1992.

[8] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, No. 1, pp. 5-35, June 1991.

[9] H. De Man, J. Rabaey, P. Six and L. J. Claesen, "Cathedral-II: A Silicon Compiler for Digital Signal Processing", *IEEE Design and Test*, Vol. 3, No. 6, pp. 13-25, Dec. 1986.

[10] Keshab K. Parhi, **VLSI Digital Signal Processing Systems: Design and Implementation**, Wiley Inter-Science, 1999.

[11] K. K. Parhi and D. G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimal Unfolding", *IEEE Trans. on Computers*, Vol. 40, No. 2, pp. 178-195, Feb. 1991.

[12] I. C. Park and C. M. Kyung, "Fast and Near Optimal Scheduling in Automatic Data Path Synthesis", *Proc. of ACM/IEEE 28th Design Automation Conference*, pp. 680-685, 1991.

[13] N. L. Passos, E. H. M. Sha and L. F. Chao, "Optimizing Synchronous Systems for Multi-dimensional Applications", *Proc. of European Design and Test Conference*, pp. 54-58, March 1995.

[14] N. L. Passos and E. H. M. Sha, "Synthesis of Multi-dimensional Applications in VHDL", *Proc. of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 530-535, Oct. 1996.

[15] N. L. Passos and E. H. M. Sha, "Achieving Full Parallelism using Multi-dimensional Retiming", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, Issue 11, pp. 1150-1163, Nov. 1996.

[16] N. L. Passos, E. H. M. Sha and L. F. Chao, "Multi-dimensional Interleaving for Synchronous Circuit Design Optimization", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, Issue 2, pp. 146-159, Feb. 1997.

[17] N. L. Passos and E. H. M. Sha, "Scheduling

of Uniform Multi-dimensional Systems under Resource Constraints", *IEEE Trans. on VLSI Systems*, Vol. 6, No. 4, pp. 719-730, Dec. 1998.

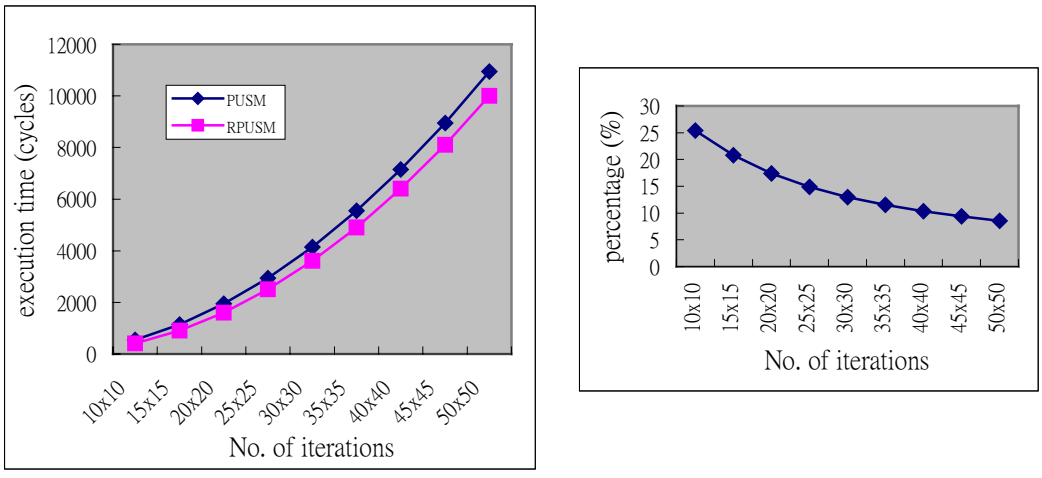
[18]N. L. Passos and E. H. M. Sha, "Synchronous Circuit Optimization via Multi-dimensional Retiming", Dept. of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN 46556.

[19]P. G. Paulin and J. P. Knight, "Forced-directed Scheduling for the Behavioral Synthesis of ASIC's", *IEEE Trans. of Computer-Aided Design*, Vol. 8, pp. 661-679, June 1989.

[20]M. L. Tsai, **A Study of Instruction Scheduling Techniques for VLIW-based DSP and Implementation of Its Simulation and Evaluation Environment**, Master Thesis, National Chiao-Tung University, June 2001.

Table 1. Characters and resource constrains for each example.

	No. of Multiplier	No. of Adder	$Length_{PUSM}$	$Length_{list}$	*Prologue*	*Epilogue*	Retiming depth
Filter 1 [14]	1	1	2	3	2	2	1
Filter 2 [16]	1	1	3	4	2	2	1
Model A [20]	1	1	4	5	6	7	2
Transmission Lines [17]	1	2	4	6	5	6	2
IIR Section [18]	2	2	4	5	4	3	1
IIR Filter [17]	2	2	4	5	8	5	2



(a) (b)

Figure 8. (a) Execution time, (b) difference percentage.

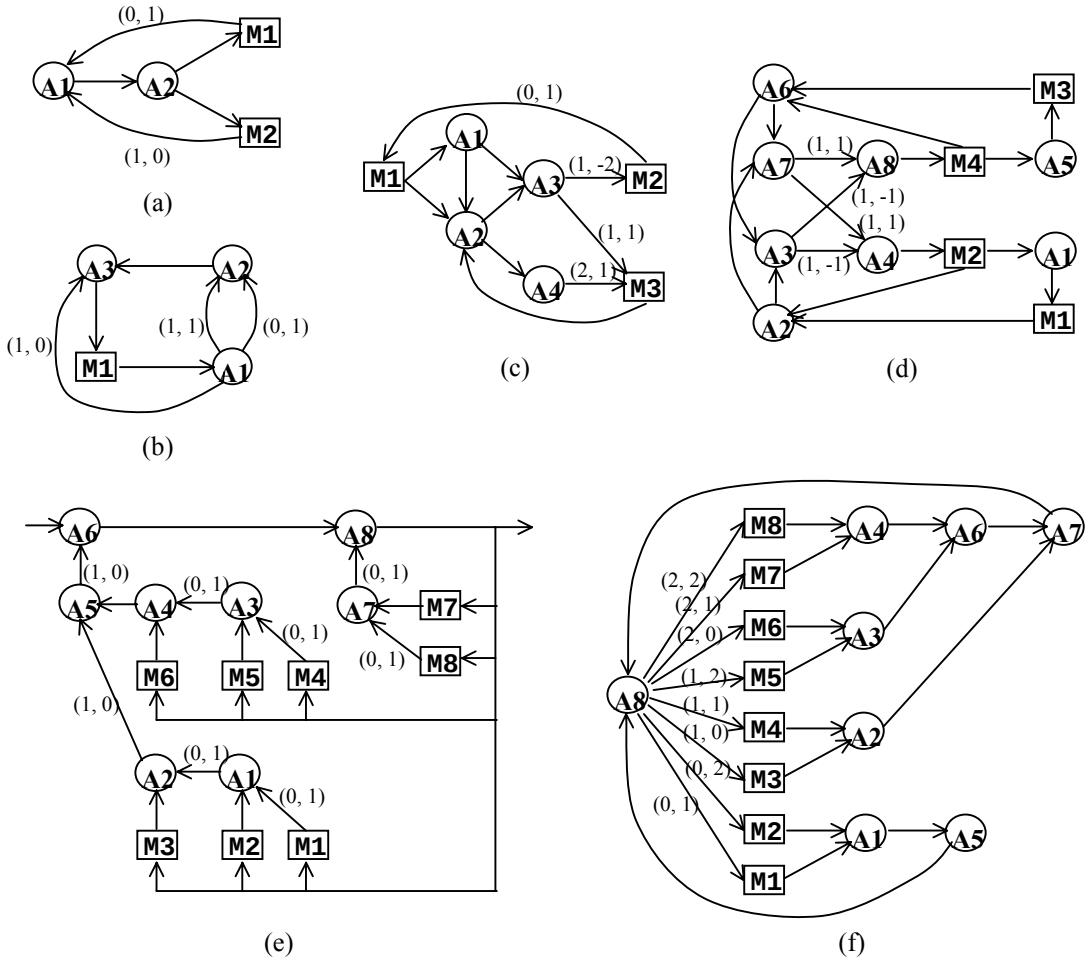


Figure 7. (a) Filter 1, (b) Filter 2, (c) Model A, (d) Transmission Lines, (e) IIR Section, (f) IIR Filter.