Increasing Parallelism of Loops with the Loop Distribution Technique[*]

Ku-Nien Chang and Chang-Biau Yang
Department of Applied Mathematics
National Sun Yat-sen University
Kaohsiung, Taiwan 804, ROC
cbyang@math.nsysu.edu.tw

Abstract

In a loop, the parallelism is bad when the statements in the loop body are involved in a data-dependence cycle. How to break data-dependence cycles is the key point for increasing the parallelism of loop execution. In this paper, we consider the data-dependence relation in the viewpoint of statements. We propose two new methods, the modified index shift method and the statement substitution-shift method. They have better parallelism and performance than the index shift method in general. The modified index shift method is obtained from modifying the index shift method and combining with the loop distribution method. The statement substitution-shift method is obtained from combining the statement substitution method, the index shift method and the unimodular transformation method with the loop distribution method. Moreover, the topological sort can be applied to determine the parallel execution order of statements.

Keywords. parallel processing, parallel complier, loop distribution, data-dependence graph

1 Introduction

For a program, as we know, if there is no cycle in the data-dependence graph, the statements can be executed parallelly and their parallel execution order can be determined by the loop distribution technique [14, 10, 9]. In this paper, our main goal is to increase the parallelism of loops that contain cycles. It is hard to break a cycle consisting of true dependence, which is a positive data-dependence cycle. We can use some methods, such as the index shift method [8] and the unimodular transformation method [1, 2, 13], to increase the parallelism of positive data-dependence cycles.

For a non-nested loop, we propose the *statement substitution-shift* method, which is the statement substitution method combined with the index shift method, to increase the parallelism of a positive data-dependence cycle. For a nested loop, we combine the statement substitution-shift method with the unimodular transformation method to increase the parallelism of a positive data-dependence cycle. Then, the statements can be executed parallelly with their parallel execution order determined by the loop distribution technique. Moreover, we also modify the index shift method [8] to get a new method, called the *modified index shift* method, which has better performance than the index shift method in general. In short words, the main goals of these new methods are all to increase the parallelism of positive data-dependence cycles.

In this paper, we assume that an unlimited number of processors are available and the calculation of an expression on the right hand side of an assignment is done parallelly in the tree-like structure. The rest of this paper is as follows. In the next section, we shall describe the program model we use in this paper and define some terminologies. In Section 3, we shall propose two new methods, the modified index shift method and the statement substitution-shift method, to increase the parallelism of non-nested loops with positive data-dependence cycles. In Section 4, we shall combine our methods with the unimodular transformation method to perform the parallelization of nested-loops. In Section 5, we give the performance analysis of our methods and the index shift method. Because the performance of these methods deeply depends on the input cases, we can not conclude which one is the best. But, we list the conditions under which our methods are superior to the index shift method. And finally, some conclusions will be given in Section 6.

[*]This research work was partially supported by the National Science Council of the Republic of China under contract NSC 84-2121-M-110-005 MS.

2 Program Model

In this section, we shall first introduce some terminologies [10, 14, 4, 12] and then define the program model we use in this paper. In a parallel program, there are three types of *DO loops*: *serial* (*DOSER*) *loops*, *do all* (*DOALL*) *loops* and *do across* (*DOACR*) *loops*. Iterations of a DOSER loop must be executed serially, iterations of a DOALL loop can be executed parallelly in any order, and in a DOACR loop, the execution on successive iterations can be partially overlapped.

In this paper, we shall only consider a *perfectly nested loop* or *one-way nested loop* with nest-depth m in the following form:

```
DO $I_1 = 1, N_1$
    DO $I_2 = 1, N_2$
        ......
        DO $I_m = 1, N_m$
            {B}
        END DO
        ......
    END DO
END DO
```

where B is a set of assignments, each updating one element of an array.

If two statements access the same data element, we say that they have *dependence* between this two statements and its dependence distance vector, or simply the dependence vector, is $J - I = (j_1 - i_1, j_2 - i_2, \ldots, j_m - i_m)$.

A *positive dependence vector* has a positive leftmost nonzero element while a *negative dependence vector* has a negative leftmost nonzero element. There are two dependence types between two statements as follows.

(1) $A \xrightarrow{+x} B$ (positive dependence): Statement B reads the new result of array A which is updated before x iterations, where $+x$ is a positive dependence vector and $+x$ may be a zero vector(true dependence).

(2) $A \xrightarrow{-x} B$ (negative dependence): Statement B reads the old data of array A which will be updated after x iterations, where $-x$ is a negative dependence vector and $-x$ may be a zero vector(antidependence).

In this paper, we only consider the most common dependence relations, true dependence and antidependence, and only consider the loops whose dependence vectors are all constant. If a dependence distance vector is not a constant vector, we can apply the *dependence uniformization technique* [11, 4] to make it be uniform.

Each loop can be represented by a *data-dependence graph*, shortened as *DDG*, $G = (V, A)$, in which each node represents a statement (an array) and each arc in A represents a dependence relation between two statements. And, each node is associated with a number which is the execution time of the corresponding statement. Moreover, each arc is associated with a *dependence type* and a *dependence distance* to represent the corresponding dependence relation. Note that an arc may be a *self-loop* (*self-dependence*), which is dependent on itself. For example, the DDG of the following loop, L1, is shown in Figure 1(a).

```
$L1$ :
    DO $I = 0, 1000$
        $A[I] = B[I-6] + E[I-1]$
        $B[I] = 5 * C[I-2] * D[I]$
        $C[I] = 9 * A[I-2]$
    END DO
```

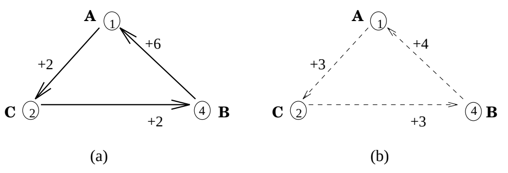

Figure 1: An example for illustrating the index shift method. (a)A positive data-dependence cycle. (b)Solving a positive data-dependence cycle by the index shift method, with the iteration window size equal to 3.

In this paper, if an arc is removed for some reason, it will be represented by a dotted arc, and its dependence relation is preserved for loop transformation. Besides, $\|A\|$ is used to denote the sum of the distances of all arcs in A.

3 Loop Transformation of Non-nested Loops

In this section, we shall discuss how to do *loop distribution* [14, 10] for a non-nested loop. It is divided into three cases. (1)The data-dependence distances in a DDG are all negative. (2)The data-dependence distances in a DDG are all positive. (3)Some of the data-dependence distances are positive and some of the them are negative. In these three cases, increasing parallelism of the positive data-dependence cycles is our main goal.

3.1 The adding temporary arrays method

The method of *adding temporary arrays*, which is called the *variable copying* method [14] or the *variable renaming* method [7] or the *node* (*statement*) *splitting* method [10, 9] in other papers, can be applied to break a cycle consisting of only negative arcs. In a negative cycle, each statement reads the old data of other arrays. If we first copy one of the arrays into a temporary array, then we can update the arrays one by one, all elements in each array being updated in parallel. If there are more than one cycle in the DDG, more than one node (array) may have to be selected to add temporary arrays.

Given a DDG with all negative arcs, to break all cycles in the DDG with minimum number of temporary arrays is equivalent to equivalent to the *feedback vertex set* problem, defined as follows.

For a directed graph $G = (V, A)$, given a positive integer $K \leq |V|$, does there exist a subset $V' \subseteq V$ with $|V'| \leq K$ such that for each cycle in G, there is at least one vertex in V'.

The feedback vertex set problem has been proved to be NP-hard [5]. Thus, the problem for adding minimum temporary arrays is also NP-hard.

3.2 The modified index shift method

When the distances of the arcs in a DDG are all positive, we can apply the index shift method, proposed by Liu et al. [8], to increase the parallelism. The DDG of L1 is shown in Figure 1(a). After applying the index shift method [8], we get the following loop, $L2$, whose DDG is shown in Figure 1(b).

```
  $L2$ :
      DOSER $I' = 0, 1001, 3$
          DOALL $I = I', min\{1001, I' + 2\}$
              PARALLELDO
$A$ :                 $A[I + 1] = B[I - 5] + E[I]$
$B$ :                 $B[I - 1] = 5 * C[I - 3] * D[I - 1]$
$C$ :                 $C[I] = 9 * A[I - 2]$
              END PARALLELDO
          END DOALL
      END DOSER
```

The parallelism in the *iteration level* of a loop is equal to the distance of the shortest arc in the DDG. Thus, the parallelisms in the iteration level of loop $L1$ and $L2$ are 2 and 3 respectively, and statements A, B and C within the same iteration can be executed simultaneously. So, it is clear that the parallelism in the *statement level* has been improved from $2 \cdot 3 = 6$ to $3 \cdot 3 = 9$. Suppose each addition operation and each multiplication operation take 1 and 2 time units respectively. The parallel execution of statements A, B, and C in this loop requires 4 time units, since there are two multiplications in statement B. Hence, the parallel execution of the loop, after the index shift method is applied, takes $\lceil 1002/3 \rceil \cdot 4 = 1336$ time units.

Definition 1 *The iteration window is an iteration space which is the whole or partial space of the iteration space of a loop.*

The DDGs of a loop may be different if we set different iteration window sizes. It is obvious that in Figure 1(a), if we set the iteration window size to be 2, then all arcs with distance greater than or equal to 2 are removed and thus no dependence arc remains. Similarly, in Figure 1(b), if we set the iteration window size to be equal to 3, then all dependence arcs will be removed. And, if the iteration window size is set to be 6 in Figure 1(a), arc (B, A) will be removed. Then, it is cycle-free in Figure 1(a). So, we can apply the index shift technique to make one arc have the maximum dependence distance, then we set the iteration window size to be this maximum dependence distance to break the cycle.

Our *modified index shift* method first shift indices to make all arcs be equal to positive zero (loop-independent dependence), except the maximum arc. After the indices are shifted, the distance of the maximum arc must be equal to the length of the cycle. Then, we set the iteration window size to the cycle length 10. The resulting cycle is shown in Figure 2 and the corresponding loop, $L3$, is as follows.

```
  $L3$ :
      DOSER $I' = -3, 1000, 10$
$A$ :     DOALL $I = I', min\{1000, I' + 9\}$
              $A[I] = B[I - 6] + E[I - 1]$
          END DOALL
$B$ :     DOALL $I = I', min\{1000, I' + 9\}$
              $C[I + 2] = 9 * A[I]$
          END DOALL
$C$ :     DOALL $I = I', min\{1000, I' + 9\}$
              $B[I + 4] = 5 * C[I + 2] * D[I + 4]$
          END DOALL
      END DOSER
```

In loop $L3$, the parallelism in the iteration level becomes 10, and loops A, C and B must be executed sequentially. So, the parallelism in the statement level is also 10. Here, we can view a cycle-

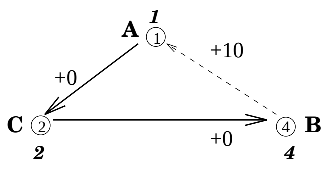

Figure 2: Solving a positive data-dependence cycle by the modified index shift method, with the iteration window size equal to 10.

free DDG as an activity on vertex(AOV) network [6]. Then, the techniques for solving problems on an AOV can be applied. Because the critical path is path ($A \rightarrow C \rightarrow B$), the critical time is $1 + 2 + 4 = 7$ and the critical path should be executed $\lceil 1004/10 \rceil = 101$ times, the parallel execution of the loop after applying our modified index shift method requires $101 \cdot 7 = 707$ time units. We must note that we cannot change the dependence type of an arc after applying the index shift technique, because some statements may read wrong data. Moreover, it is called *valid index shift operations.*

Furthermore, we can apply the loop fusion method [9, 14] to reduce start-up costs for DOALL loop, and increase overlapping of their synchronizations.

Definition 2 *A π-block is defined as a strongly connected component of a DDG in which all negative arcs are removed. Note that a π-block must contain at least one cycle which is not a self-loop.*

If there is more one cycle in the same π-block, we have to shift the indices such that the number of arcs with positive zero is maximized. Our modified index shift method is as follows.

```
Algorithm 1 Modified Index Shift (MIS)
  Input : A data-dependence graph $G = (V, A)$.
  Output: The iteration window size $W_j$, $1 \leq j \leq r$, where $r$ is the number of $\pi$-blocks, and a cycle-free data-dependence graph $G' = (V, A')$, $A' \subseteq A$.
    Step 1: For every $\pi_j$ in $G$ do Step 2 through Step 10.
    Step 2: Calculate the iteration window size $W_j$, which is equal to the length of the positive data-dependence cycle with the minimum length in $\pi_j$.
    Step 3: Let $K = \infty$ and $W'_j = W_j$.
    Step 4: Calculate $R_i = \lfloor \|C_i\| / W'_j \rfloor$ for each positive data-dependence cycle $C_i$ in $\pi_j$.
    Step 5: By the descending order of $\|C_i\|$'s, remove $R_i$ arcs to break each positive data-dependence cycle $C_i$ in $G$ such that the critical time $CR$ of this strongly connected component $\pi_j$ is minimized. Let $I'$ denote the set of the selected arcs.
    Step 6: If the number of cycles in $\pi_j$ is 1, then $I = I'$ and goto Step 9.
    Step 7: Calculate the average execution time of one iteration in $\pi_j$, $K' = CR/W_j$. If $K' \leq K$ then $K = K'$, $I = I'$ and $W_j = W'_j$; otherwise go to Step 9.
    Step 8: If $|I'|$ is not equal to the number of arcs in $\pi_j$, then $W'_j = W'_j - 1$, restore all removed arcs in $I'$ to their original cycles and go to Step 4.
    Step 9: Shift the indices such that the length of each arc in set $I$ is greater than or equal to $W_j$, and no arc changes its sign (dependence type). Note that if there is more than one cycle in the same $\pi$-block, we have to shift the indices such that the number of arcs with positive zero is maximized under the condition which does not influence the parallelism in iteration level.
    Step 10: Delete the positive arcs in set $I$ from the DDG.
end
```

In Algorithm MIS, because the values of $R_i \cdot W_j$ must be not greater than $\|C_i\|$ for each positive cycle C_i in G. By [8], we can perform a sequence of valid index shift operations such that each arc in I is greater than or equal to W and all arcs not in I are greater than or equal to zero.

See [3] for a more complicated example.

3.3 The statement substitution-shift method

Our *statement substitution-shift* method is divided into two phases: (1), *minimum statement substitution* phase: to choose the proper statements for statement substitution; (2). *s-statement index shift* phase: to shift the indices to get better parallelism in the iteration level. If only one node in a π-block is chosen to do statement substitution, we only perform phase 1. Otherwise, we should perform phase 2 following phase 1.

Now, we apply our statement substitution-shift method to solve loop $L1$. If we do the substitution on statement A, then the loop becomes the following loop, $L4$, which contains loops $A1, A2, A3$, C and B.

```
      L4 :
          PARALLELDO
A1 :        DOALL I = 1, 6
              A[I] = B[I - 6] + E[I - 1]
            END DOALL
A2 :        DOALL I = 7, 8
              A[I] = 5 * C[I - 8] * D[I - 6] + E[I - 1]
            END DOALL
          END PARALLELDO
A3 :        DOSER I' = 9, 1000, 10
              DOALL I = I', min{1000, I' + 9}
                A[I] = 5*9*A[I-10]*D[I-6]+E[I-1]
              END DOALL
            END DOSER
C  :        DOALL I=1, 1000
              C[I] = 9 * A[I - 2]
            END DOALL
B  :        DOALL I=1, 1000
              B[I] = 5 * C[I - 2] * D[I]
            END DOALL
```

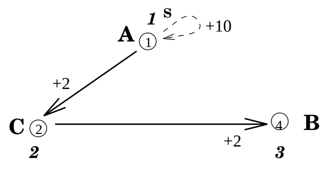

Figure 3: An example for the statement substitution-shift method.

The resulting DDG is shown in Figure 3, in which a node marked with `S` means that statement substitution is applied on that node and all incoming positive arcs are removed. The parallelism of loop A in the iteration level is 10. Note that we apply the topological sort [6] to determine the order of parallel execution.

If the number of available processors is unlimited, then all loops, except loop $A3$, have the maximum parallelism 1000. The parallel execution of the statement $A[I] = 5*C[I-8]*D[I-6]+E[I-1]$ and the statement $A[I] = 5*9*A[I-10]*D[I-6] + E[I-1]$ take 5 and 5 time units respectively, so the parallel execution of loop $L4$ requires $5 + \lceil 992/10 \rceil \cdot 5 + 2 + 4 = 511$ time units. Note that the parallel execution of a statement can be done in a tree-like style.

Proposition 1 *Let m_i denote the minimum number of nodes in cycle C_i that are selected to do statement substitution to break all original positive data-dependence cycles. The parallelism in the iteration level of the π-block containing C_i is at least $M = \min_{1 \le i \le r}\{\lfloor \|C_i\|/m_i \rfloor\}$, where r is the number of positive data-dependence cycles in this π-block. Because $M \cdot m_i$ is not greater than $\|C_i\|$ in this π-block, we can perform a sequence of valid index shift operations.*

Proposition 2 *In positive data-dependence cycle C_i, let S_i denote the maximum number of nodes that are selected to do statement substitution under the condition which does not decrease the parallelism of the π-block containing C_i. Then, S_i is equal to $\lfloor \|C_i\|/M \rfloor$, where M is calculated by Proposition 1.*

Algorithm 2 *Minimum Statement Substitution(*MSS)

Input : A data-dependence graph G.

Output: A data-dependence graph G' in which there is no cycle, except self-loops or the cycles consisting of the s-statements only.

Step 1: Break all original positive cycles of G by choosing a set of nodes with minimum size to do statement substitution. If there are more than one set having the minimum size, choose the set such that for the nodes in the set, the total number of positive incoming arcs contained in some positive cycles is minimized.

Step 2: For every π-block, if there is only one cycle, go to Step 4.

Step 3: For every positive cycle C_i, try to do statement substitution on a set of nodes whose size is less than or equal to S_i. Note that the value of S_i can be calculated by Proposition 2.

Step 4: In G, mark all nodes chosen in the above steps with `S`, and construct the new arcs connected from their end-nodes, and delete their original positive incoming arcs.

end

Algorithm 3 *S-Statement Index shift(*SSIS)

Input : A data-dependence graph G.

Output: A data-dependence graph G' whose parallelism is greater than the parallelism of G.

Step 1: For every π-block, if there are more than one node marked with `S`, do Step 2.

Step 2: Apply the index shift technique such that the distances of the new arcs pointing to the s-statements are greater than or equal to M. Note that the value of M is calculated by Proposition 1.

end

In Step 1 of Algorithm MSS, we want to select a subset of vertices with minimum size such that there is no positive data-dependence cycle remaining in the graph after the removal of the selected vertices. This problem is equivalent to the feedback vertex set problem, which has been proved to be NP-hard [5]. Thus, Step 1 of Algorithm MSS is an NP-hard problem.

In fact, we can view the nodes that are marked with `S` in the same π-block as a single super node, then there is no cycle in the DDG except self-loops after applying the statement substitution phase.

3.4 General loop transformation

In general, in a loop, the data-dependence distance may be positive or negative. Because the adding temporary arrays method produces the maximum parallelism with little overhead, we apply the adding temporary arrays method to break all data-dependence cycles except positive data-dependence cycles. Then, we break positive data-dependence cycles. For details, refer to [3].

4 Loop Transformation of Nested loops

The parallelization of nested loop can be easily handled by the expansion of parallelizing non-nested loops. In a nested loop, there are also three types of data-dependence cycles: negative cycle, positive cycle and mixed cycle. We can also apply the adding temporary arrays method to break the negative cycles in a mixed DDG. Therefore, for solving nested loops, we still focus on how to increase the parallelism of a positive data-dependence cycle.

For a nested loop, the index shift method proposed by Liu et al. [8] uses the hyper-plane method to improve its parallelism. It transforms some dimensions of all dependence vectors into one dimension, then shifting proper indices is done in the same way that in a non-nested loop. Naturally, we may use the modified index shift method instead of the index shift method.

A nested loop can be viewed as several non-nested loops. As mentioned in the previous section, a positive data-dependence cycle of a non-nested loop can be executed partially parallelly by setting an appropriate iteration window size. Similarly, for a nested loop, after applying the statement substitution-shift method, we try to sacrifice one of the dimensions to do partial parallelization (do across), which breaks all positive self-loops by setting an appropriate iteration window size. Then other dimensions of the nested loop can be perfectly parallelized (do all).

Sometimes, to transform a nested loop, we can apply the *unimodular transformation* method [1, 2, 13], which parallelizes one dimension perfectly and parallelize other dimensions partially. The unimodular transformation method increases a large amount of parallelism in general with little overhead in arithmetic computation of the loop and calculation of statement subscriptions. Hence, if we can't find one dimension of the nested loop to be partially parallelized (to break all positive self-loops), we will apply the *unimodular transformation* method to transform the original loops. Figure 4 is used to illustrate our method.

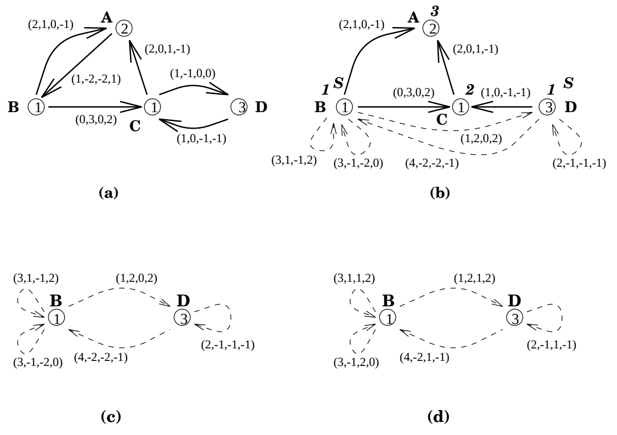

Figure 4: The DDGs for illustrating the transformation of nested loops. (a) The original DDG. (b) The DDG after applying Algorithm MSS. (c) The simplified DDG from (b). (d) The DDG after applying the index shift method.

Figure 4(a) shows the original DDG. There are three cycles $C_1 = \{A, B, C\}$, $C_2 = \{A, B\}$ and $C_3 = \{C, D\}$, and $\|C_1\| = (3, 1, -1, 2)$, $\|C_2\| = (3, -1, -2, 0)$, $\|C_3\| = (2, -1, -1, -1)$. We get Figure 4(b) after applying Algorithm MSS, which breaks the original cycles C_1, C_2 and C_3 but produces new self-loops on statements B and D with length $\|C_1\|, \|C_2\|$ and $\|C_3\|$. Therefore, we will focus on how to increase parallelism on these self-loops.

Consider the individual dimensions of dependence vectors $(3, 1, -1, 2)$, $(2, -1, -2, 0)$ and

$(3, -1, -1, -1)$. Only dimensions I_1 and I_3 have the same sign and all of these dependence vectors are greater than zero. Thus, for statements B and D, we can perform partial parallelization on the first level (dimension) or the third level (dimension) of the nested loop, and perform perfect parallelization in other levels (dimensions). If we choose the first level of the nested loop for performing partial parallelization, the parallelism is 2 on this level. On the contrary, if we choose the third level of the nested loop, the parallelism is 1 on this level. In fact, the parallelism in the iteration level not only depends on the above parallelism, but also depends on the upper bound of the loop index. For example, suppose that the upper bounds of I_1 and I_3 are 200 and 50 respectively. Then, the parallel execution of statement B and statement D takes $200/2 = 100$ steps if the partial parallelization is performed on the first level, and $50/1 = 50$ steps if the partial parallelization is performed on the third level. Hence, we should choose the third level of the nested loop for performing partial parallelization.

If we ignore all nodes except node B and node D, we can simplify Figure 4(b) as Figure 4(c). In Figure 4(c), consider dimension I_3 of dependence vectors $(4, -2, -2, -1)$ and $(1, 2, 0, 2)$. We can apply the index shift technique to increase parallelism in the iteration level, as shown in Figure 4(d).

5 Performance Analysis

We first list the parameters of the DDG which are used in this paper.
(1)N: the number of iterations in the loop.
(2)π_j: the j-th strongly connected component of the DDG.
(3)C_i: the i-th cycle of some π-block.
(4)C_m: the cycle whose length is minimum in some π-block.
(5)n_i: the number of statements in cycle C_i.
(6)ρ_j: the length of cycle C_m in π_j.
(7)z_i: the length of arc with maximum length in cycle C_i.
(8)AM_i: the arithmetic mean of all arcs' lengths in cycle C_i. In other words, $AM_i = \lfloor \|C_i\|/n_i \rfloor$, for any C_i.
(9)AM_m: the minimum arithmetic mean of all cycles. In other words, $AM_m = \underset{1\le i\le r}{min}\{AM_i\}$, where r is the number of cycles in the DDG.
(10)t: the maximum execution time of all statements in the DDG.
(11)T_i: the execution time of cycle C_i for one iteration. It implies that $T_i \le n_i \cdot t$.
(12)W_j: the iteration window size for π_j of the DDG.
(13)R_i: the number that is equal to $\lfloor \|C_i\|/W_j \rfloor$, for cycle C_i in π_j.
(14)CT_j: the parallel execution time of π_j for one iteration. In other words, $CT_j \le \underset{1\le i\le r}{max}\{T_i\}$, where r is the number of cycles in π_j.

Given a loop L, let α and β be the total execution time of parallel loops produced by the index shift algorithm [8] and Algorithm MIS respectively. We have following propositions.

Proposition 3 *$\alpha = \lceil (N + c)/AM_m \rceil \cdot t$, where c is a constant and $c \le 2 \cdot AM_m$. Moreover, the value of c is dependent on the permutation of dependence distances in all cycles.*

Proposition 4 *$\beta = \sum_{j=1}^{r} \lceil (N + c_j)/W_j \rceil \cdot CT_j$, where r is the number of π-blocks in the DDG. The value of each c_j, $1 \le j \le r$, is dependent on the connection conditions of cycles in the π-block. The value of each CT_j, $1 \le j \le r$, is also dependent on the permutation of the execution times of the nodes in the π-block. In Algorithm MIS, $CT_j \le \underset{1\le i\le q}{max}\{T_i/R_i\}$ where q is the number of cycles in π_j.*

Lemma 1 *If there is only one positive cycle in the DDG, $\beta = \lceil (N + \rho - z)/\rho \rceil \cdot T$.*

Proof: See [3].

Now, we have the following theorems.

Theorem 1 *If there is only one positive cycle in the DDG and $\rho \le \frac{z \cdot T}{1+T-2\cdot t}$, then $\beta \le \alpha$.*

Proof: See [3].

If there is more than one positive data-dependence cycle in the DDG, then $\beta \le \alpha$ because Algorithm MIS includes the index shift algorithm [8].

Theorem 2 *Given a non-nested loop L, if there is only one positive cycle in every π_i of the DDG, then $\gamma \le \beta$, where γ is the execution time of the node marked with* S *by Algorithm MSS.*

Proof: See [3].

6 Conclusions

By the analysis in the previous section, we get that the modified index shift method does not always have better performance than the index shift method [8]. However, the modified index shift method improves parallelism much more than the index shift method in most cases, and thus gets better performance than the index shift method in general. In the following conditions, the modified index shift method will get better performance.

(1) The number of strongly connected components is few.

(2) The lengths of the cycles with minimal length are different in every strongly connected components, and their differences are sufficiently large.

(3) The difference between the maximum execution time and the average execution time of all nodes (statements) is large in the DDG.

(4) The length of the minimum length cycle in the DDG is very small and the node with maximum execution time of nodes in the DDG is very large.

If the parallelism in the iteration level of one method is better, the performance of vectorization obtained from it is also better. Thus, in general, our modified index shift method has better performance in vectorization.

In general, if the depth of parenthesis nesting in each statement is small or the length of each expression is short or the number of strongly connected components is large, we should apply the statement substitution-shift method, instead of the modified index shift method, to get good performance. For a nested loop, we first apply the statement substitution-shift method to decrease the number of influential dependence vectors (only the dependence vectors of self-loops are remained), then apply the unimodular transformation method to transform the nested loops. Hence the parallelism of our method is better than that obtained by directly applying the unimodular transformation method.

References

[1] U. Banerjee, *Loop Transformations for Restructuring Compilers:The Foundations.* Boston,Dordrecht,London: Kluwer Academic Publishers, 1993.

[2] U. Banerjee, *Loop Transformations for Restructuring Compilers:The Advanced.* Boston,Dordrecht,London: Kluwer Academic Publishers, 1995.

[3] K. N. Chang, *Increasing Parallelism of Loops with the Loop Distribution Technique.* Master Thesis, Department of Applied Mathematics, National Sun Yat-sen University, Kaohsiung, Taiwan, 1995.

[4] D. K. Chen, *Compiler Optimizations for Parallel Loops with Fine-Grained Synchronization.* Ph.D. dissertation, Dep. Comput. Sci. Univ. Illinois at Urbana-Champaign, 1994.

[5] M. R. Garey, *Computers and Intracrability: A Guide to Theory of NP-completeness.* San Francisco, USA: Freeman, 1979.

[6] E. Horowitz and S. Sahni, *Fundamentals of data structures in Pascal.* Computer Science Press, 4 ed., 1994.

[7] D. J. Lilja, “Exploiting the parallelism available in loops,” *c*, pp. 13–26, Feb. 1994.

[8] L. S. Liu, C. W. Ho, and J. P. Sheu, “Synthesizing the parallelism of nested loops using an index shift method,” *Journal of Information Science and Engineering*, Vol. 7, pp. 529–541, 1991.

[9] D. Padua and M. Wolfe, “Advanced compiler optimizations for supercomputers,” *Communications of the ACM*, Vol. 29, No. 12, pp. 1184–1201, Dec. 1986.

[10] C. D. Polychronopoulos, *Parallel Programming and Compilers.* Assinippi park, norwell, massachusetts,USA: Kluwer Academic, 1 ed., 1988.

[11] T. H. Tzen and L. M. Ni, “Dependence uniformization: A loop parallelization technique,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 5, pp. 547–558, May 1993.

[12] C. M. Wang and S. D. Wang, “A hybrid scheme for efficiently executing nested loops on multiprocessors,” *Parallel Computing*, Vol. 18, pp. 625–637, 1992.

[13] M. E. Wolf and M. S. Lam, “A loop transformation theory and an algorithm to maximize parallelism,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452–471, Oct. 1991.

[14] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers.* ACM press, 1 ed., 1991.