ソフトウェア工学　88－7
(1992. 11. 10)

並列化多重ループにおける冗長な同期を解消する手法

李 暁傑　　原田 賢一
li@hara.cs.keio.ac.jp　　harada@hara.cs.keio.ac.jp

慶應義塾大学大学院 理工学研究科 計算機科学専攻
〒223　横浜市　港北区　日吉 3-14-1

あらまし

概　要

スーパーコンピュータの性能を十分に活すために、並列化コンパイラの研究が精力的に進められている。一般に、プログラム中のループには最も高い並列性が存在する。ループを中心とした並列化手法として、繰返し間での同期をとることによって、ループの並列化を図るために、**Doacross** と呼ばれる同期機構が提案されている。しかし、高い並列化を得るためだけに多くの同期を導入すると、冗長な同期コードが生成されるという欠点がある。一重ループの同期は規則性を保つが、多重ループの場合には、同期の規則性が失われる。本論文では、多重同期空間における冗長な同期を解消する方法を示す。

和文キーワード　　並列化多重ループ、冗長な同期、最適化、コンパイラ、並行プログラミング

Removing Redundant Synchronizations from N-nested Parallel Loops

Shiao-Chieh LEE　　Ken'ichi HARADA
li@hara.cs.keio.ac.jp　　harada@hara.cs.keio.ac.jp

Department of Computer Science, Graduate School of Science and Technology,
Keio University
3-14-1, Hiyosi, Kouhoku-ku, Yokohama, 223, Japan

Abstract

Abstract

Data dependences constrain the parallel execution of an imperative program and are typically held by using synchronization instructions. Execution of these instructions occupies a significant part of the overhead in the parallel program. Some of the data dependences in the program may be redundant because they are covered by some other dependences. In a simple loop, if a synchronization is redundant at one iteration, it is redundant at all iterations of the loop. However, in nested loops, redundancy is not uniform. In this paper, we give a scheme of removing redundant synchronizations from n-nested parallel loops.

英文 key words　N-nested parallel loop, Redundant synchronization, Optimization, Compiler, Concurrent programming

1 Introduction

A major problem with programming for multiprocessor has been that it is difficult to make full use of the inherent parallelism. One solution to this problem is the development of software systems providing programming environments and languages that free the user from the details of the architecture. A key component of such systems is a parallel compiler. Several language constructs such as **Doacross** and **Doall**[1] have been designed to control the parallel execution of loops. When a loop is executed in parallel on a multiprocessor, synchronization is needed to hold data dependences between statements executed in different iterations of the loop by different processors. Techniques to compile loops for execution on multiprocessors were developed in [2],[3].

Data dependences (simply called as *depedences*, below) constrain the parallel execution of programs. The dependence structure of a computation can be characterized by a directed acyclic graph (DAG), where the nodes represent tasks and directed edges represent precedence constraints.These dependence edges correspond to explicit synchronization required under asynchronous model of parallel execution. If there is an alternate path from the source node of a dependence to the sink node, then synchronization instruction corresponding to the dependence from source to sink is superfluous because the occurrence of the synchronization event through the alternate path implies that the precedence constrains implied by this edge is implicitly guaranteed.

The problem of recognizing and removing redundant dependences in simple loops with constant dependences is studied by Midkiff and Padua [3]. Their algorithm searchs for a sequence of dependences that can cover a given dependence. However, they only showed removal of redundant synchronization from simple loops, in fact, n-nested loop is often used in the scientific computation. In this paper, we discuss the method for detecting redundant synchronizations from parallel loops with a new algorithm proposed by us in [4].

2 Removing of redundant synchronizations from nested loops

Because the direction of the synchronization is inconsist in n-nested loops. The representation of redundant synchronizations becomes more difficult. In this section, we disscus how to charactrize the redundant synchronizations, and to show the conditions of deleting these synchronizations.

At first, we define the n-nested loop as follows.

```
    DO $i_1 = L_1, U_1$
      DO $i_2 = L_2, U_2$
        ...
        DO $i_n = L_n, U_n$
S1        A($i_1+k_1$, $i_2+k_2$,...,$i_n+k_n$)= ...
S2        ... =A($i_1+h_1$, $i_2+h_2$,...,$i_n+h_n$)
        ENDDO
        ...
      ENDDO
    ENDDO
```

In this loop we say that there is a dependence lock $<(i_1+k_1, i_2+k_2, ..., i_n+k_n), (i_1+h_1, i_2+h_2, ..., i_n+h_n)>$, between S1 and S2. This dependence lock is represented as a dependence in the iteration space. Iterations of a simple loop can be mapped onto discrete points on a line. Hence, a simple loop with n iterations is represented by points 1 to n on a line. Similarly, the iterations of an n-dimensional nested loop can be mapped onto discrete points in an n-dimensional space. Based on the shape of the iteration space, nested loops are classified into three types: *regular* loop, *triangle* loop, and *irregular* loop. An example of each of the three types is given in Fig.1. In the case of a regular loop the number of iterations in the inner loop is independent of the outer loop iteration. If the number of the iterations in the inner loop is a linear function of the iteration number of the outer loop, then such loops are classified as triangle. In an irregular loop the number of iterations in the inner loop is a non-linear function of the iteration number of the outer loop.

```
DO $i_1 = 1, n_1$
  DO $i_2 = 1, n_2$
    DO $i_3 = 1, n_3$
      ...
    ENDDO
  ENDDO
ENDDO
```

a) Regular loop

```
DO $i_1 = 1, n_1$
  DO $i_2 = 1, a * i_1 + b$
    DO $i_3 = 1, c * i_1 + d * i_2 + e$
      ...
    ENDDO
  ENDDO
```

```
ENDDO
  b) Triangle loop

DO $i_1$ = 1, $n_1$
  DO $i_2$ = 1, $b[i_1]$
    DO $i_3$ = 1, $c[i_1]+d[i_2]$
      ...
    ENDDO
  ENDDO
ENDDO
  c) Irregular loop
```

Figure 1: Types of nested loop

When we view nested loops as points in an n-dimensional space, The *index vector* of an iteration gives the coordinates of the corresponding point in the discrete Cartesian coordinate system. A point in an iteration space can be viewed as a vector defined by the line connecting the origin of the coordinate system to the point. We use the notation p_i to denote a point and P_i to denote the corresponding vector. The dependence distance of a iteration dependence is given by subtracting the iteration vector of the source of a dependence from that of the sink. Hence, a dependence distance of a nested loop is a vector in n-dimensional space. The same as a simple loop, for any point p_1, and dependence distance vector d_i, the adjacent point of p_1 due to the dependence is given by vector of $P_1 + d_i$.

Without loss of generality, we can assume that the lower bound of every loop control variables of loops is one. First, we will consider loops that have regular iteration space, that is upper bound of iteration are independent loop. In the next, we define dependence distance vector.

```
DO I = 1,5
  DO J = 1,5
    A(I, J) = ...
    ... = A(I, J+3)
    ... = A(I+1, J-4)
    ... = A(I+1, J-1)
  ENDDO
ENDDO
```

In this loop, there are three dependence locks. Dependence distance vector $(0,3)^T$ is respect to flow dependence lock $<(I,J),(I,J+3)>$, $(1,-4)^T$ is respect to dependence lock $<(I,J),(I+1,J-4)>$, and $(1,-1)^T$ is respect to dependence lock $<(I,J),(I+1,J-1)>$. In generall, an n-dimensional dependence distance is a n-dimensional vector of subtracting dependence source from dependence sink. For example, corresponding to the dependence look $<(i_1,i_2,\ldots,i_n),(i_1+h_1,i_2+h_2,\ldots,i_n+h_n)>$, the dependence distance vector is $(h_1,h_2,\ldots,h_n)$. Because, the dependence distance vector represents the relation of a dependence lock, we say dependence distance vector as dependence vector for short.

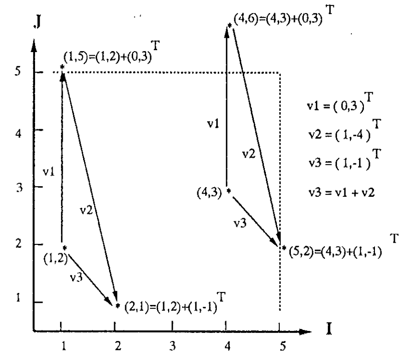

Figure 2: Dependence of n-nested loop

Unlike simple loops, redundancy of a dependence is not uniform in the iteration space of n-dimensional space. This means that if a dependence vector was redundant at one point in the iteration space, we can not conclude that this dependence vector is redundant at each point in the iteration space. We show this fact in the Fig.2. In this situation, $(1,-1)^T$ is redundant at point $(1,2)$, but it is not redundant at point $(4,3)$.

Before we define the redundancy of dependence in n-nested loops, the bound of loop, *bottom,top* are defined as follows:

Definition 1. The relation between two points in the iteration space, $\trianglelefteq$ and $\trianglerighteq$ are defined as:
• The relation $p_1 \trianglelefteq p_2$ between two points p_1 and p_2 can be obtained, if and only if $P_{1j} \le P_{2j}$ $(1 \le j \le n)$, where $(P_{11},P_{12},\ldots,P_{1n})$ and $(P_{21},P_{22},\ldots,P_{2n})$ are vectors representing points p_1 and p_2, respectively.
• The relation $p_1 \trianglerighteq p_2$ between two points p_1 and p_2 can be obtained, if and only if $P_{1j} \ge P_{2j}$ $(1 \le j \le n)$, where $(P_{11},P_{12},\ldots,P_{1n})$ and $(P_{21},P_{22},\ldots,P_{2n})$ are vectors representing points p_1 and p_2, respectively.

The *bottom* and *top* are these points such that each p in the iteration space can be characterized by $p \trianglerighteq$ *bottom* and $p \trianglelefteq$ *top*. *i.e. bottom* and *top* respectively be the points representing the first and last iterations of a regular loop.

We define a redundant dependence of a n-nested loop as follows.

Definition 2. Let $\Delta = \{d_1, d_2, ..., d_m\}$ be the set of dependence vectors. An outgoing edge due to dependence d_i, $1 \leq i \leq m$ is redundant at point p_1, if and only if

1. $d_i = d_{i1} + d_{i2} + ... + d_{in}$, where $d_{ij} \in \Delta$ and $d_i \neq d_{ij}$ for all $1 \leq j \leq n$.
2. $P_1 + \sum_{k=1}^{j} d_{ik} \trianglerighteq$ *bottom* for all $1 \leq j \leq n$.
3. $P_1 + \sum_{k=1}^{j} d_{ik} \trianglelefteq$ *top* for all $1 \leq j \leq n$.

The first condition states that there is an alternate path in a regular graph of unbounded size. The second and third condition guarantee that the path is in the bounded iteration space. Let us consider the dependence vectors $(0,3)^T, (1,-4)^T, (1,-1)^T$. The dependence vector $(1,-1)^T$ is redundant in an unbounded iteration space. However, it is not always redundant at each point in the bounded iteration space.

Now we claim that if all the components of any dependence vector are non-negtive and if a dependence vector is redundant at any point then it is redundant at every other point.

Theorem 1. If none of the dependence vector of a n-nested regular loop has negtive components then the redundancy of a dependence vector at any point of the S_ISDG implies its redundancy at all points of the S_ISDG.

Proof: Let us assume that the dependence d_i is redundant at point p_1. Let p_2 be an arbitray point. Let P_1 and P_2 be the vetors representing the points p_1 and p_2, respectively.

Case 1: $P_2 + d_i \trianglerighteq$ *top*

This implies that the edge corresponding to the dependence d_i does not exist at point p_2. Hence, the theorem is obviously true.

Case 2: $P_2 + d_i \trianglelefteq$ *bottom*

Because all the components of any d_{ij} are positive, it is not possible to appear. Hence, the theorem is obviously true.

Case 3: *bottom* $\trianglelefteq P_2 + d_i \trianglelefteq$ *top*

Let dependence $d_i = d_{i1} + d_{i2} + ... + d_{in}$ at point p_1 be redundant. Let $s_j = \sum_{k=1}^{j} d_{ik}$, $1 \leq j \leq n$. Since all the components of any d_{ij} are positive, $s_p \trianglelefteq s_q$, $(1 \leq p \leq q \leq n)$ is satisfied. Because $p_2 \trianglelefteq P_2 + s_j$, $(1 \leq j \leq n)$ and $p_2 \trianglerighteq$ *bottom*, condition two is true at every point. Because $P_2 + s_n \trianglelefteq$ *top* and $P_2 + s_j \trianglelefteq P_2 + s_n$ $(1 \leq j \leq n)$, condition three is true at every point. Hence,the theorem is true. □

However, when the component of dependence vector is negtive, It is more difficult to delete redundant synchronizations from n-nested loops. Because redundancy is not uniform at every points in the iteration space. We have showed an example at Fig. 2. Since adjacent point is beyond the bound of the iteration space at some points, even if one synchronization can be deleted from node 1, it can not be always deleted from any point in the iteration space.

Without loss of generality, the 2-dimensional iteration space with negtive components will be discussed. In this situation, since a redundancy is not uniform, we give the concept of local optimization of synchronization. It means that when code optimization is begin done, a redundant synchronization code is deleted from some area of the iteration space. We say this optimizing technique as *local optimization of synchronization*. It is important that how to decide the range of optimizing area. At first, a pattern of synchronization with negtive components in dependence vector is given in Fig.3.

```
DO I = 1, $IN$
  DO J = 1, $JN$
      ... (-1,-1)$^T$
      ... (0,-2)$^T$
      ... (3,4)$^T$
      ... (2,1)$^T$
  ENDDO
ENDDO
```

The dependence vectors are (-1,-1)^T, (0,-2)^T, (3,4)^T and (2,1)^T here.

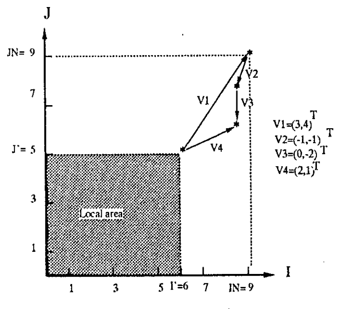

Figure 3: A pattern of negtive components

In Fig.3, we found that in the iteration space $1 \leq i \leq I'$ and $1 \leq j \leq J'$, a redundant synchronization can be deleted at any point, because adjacent points of any point are in the 2-nested loop iteration space. For analysing area bound, some definitions of symbols are given as follows. Let Δ be the set of dependence vetors, d_k be a dependence vector, $d_k = d_{k1} + d_{k2} + \ldots + d_{kn}$ and $d_{kl} = (\ i_{kl},\ j_{kl}\ )$. We difine im and jm are the sum of positive distance of dependence vectors, in and jn are the sum of negtive distance of dependence vectors.

$$im = \textstyle\sum_{l=1}^{n} i_{kl},\ i_{kl} > 0\ ,$$
$$jm = \textstyle\sum_{l=1}^{n} j_{kl},\ j_{kl} > 0\ ,$$
$$in = \textstyle\sum_{l=1}^{n} |i_{kl}|,\ i_{kl} < 0\ ,$$
$$jn = \textstyle\sum_{l=1}^{n} |j_{kl}|,\ j_{kl} < 0\ ,$$

Next, we present a sufficient condition for the bound of uniform redundancy.

Theorem 2. Let im and jm be the longest positive distances of dependence vector, in and jn be the longest negtive distances of dependence vetor, $top{=}(IN, JN)$. If $im > in$ and $jm > jn$, in the iteration space $1 \leq I \leq IN - im$, $1 \leq J \leq JN - jm$, redundancy can be deleted from any point.

Proof: Let dependence $d_i = d_{i1} + d_{i2} + \ldots + d_{in}$ at point p_1 be redundant. $s_j = \sum_{k=1}^{j} d_{ik}$ $(1 \leq k \leq n)$ and p_2 be the point in the local area. we divide the $\{\ d_{i1}, d_{i2}, \ldots\ , d_{in}\ \}$ to two parts such that in the first part, $\{\ d_{i1}, d_{i2}, \ldots\ , d_{il}\ \}$ are positive, in the second part, $\{\ d_{il+1}, d_{il+2}, \ldots\ , d_{in}\ \}$ are negtive. s_i is such the pass, which consists of $d_{i1} + d_{i2} + \ldots + dil$ at first, then $s_i = s_i + d_{il+1} + d_{il+2} + \ldots + d_{in}$. Beause p_2 in local area, $\sum_{j=1}^{l}$ i of $d_{ij} \leq im$ and $\sum_{j=1}^{l}$ j of $d_{ij} \leq jm$, $bottom \trianglelefteq P_2 + d_i \trianglelefteq top$. Because $im > in$ and $jm > jn$, $0 \leq \sum_{j=1}^{n}$ i of d_{ij} and $0 \leq \sum_{j=1}^{n}$ j of d_{ij} are true, $bottom \trianglelefteq P_2 + \sum_{i=1}^{n} s_i \trianglelefteq top$. Hence, redundancy is uniform at every point in the local area. □

Generally, when some of the components of dependence vectors are negative, according to the Theorem 2, we can delete the redundant dependence from a local area. Next we present a sufficient condition for the uniformity of a redundancy in a bounded S_ISDG. The following notation is used in presenting it. A dependence vector d_i is denoted by its components $(C_{i1}, C_{i2})^T$. Let, $\Delta = \{d_1, d_2, \ldots, d_m\}$ be the set of dependence vectors. We define p_{max} as $Max(Max(c_{i2} | 1 \leq i \leq m), 0)$. That is, p_{max} is equal to zero if the second component of all the dependences. Similarly, n_{min} is defined as $Min(Min(\{c_{i2} | 1 \leq i \leq m\}), 0)$. That is, n_{min} is equal to zero if the second component of all the dependence vectors is positive, otherwise, it is the minimun of the second component of all the dependences. We define an_{min} as the absolute value of a n_{min}. In a doubly nested loop is greater than or equal to the sum of p_{max} and an_{min}, then redundancy is uniform throughout the iteration space. Before proving this, we prove the following lemma.

Lemma 1. Let p_1 be a point in a regular rectangular iteration space of width $p_{max} + an_{min}$ and $s_i = \sum_{k=1}^{n} e_{ik}$, where $n \geq 1$ and $e_{ik} \in \Delta$ not necessarily distinct. From any point within the rectangular grid, if the adjacent point due to s_i is within the grid, then there exists a path of length n given by a sequence of adjacent points in the grid that corresponds to some permutation of $e'_{ik}s$.

Proof: We prove this using induction on the length of the path, that is the number of $e'_{ik}s$. Let p_2 be the point given by $P_1 + s_i$.
Basis: When k is equal to one, s_i is equal to e_{i1}. Since k is equal to one, there is only one permutation and the path is given by p_1, p_2.
Hypothesis: Let us assume that for any $s_i = \sum_{k=1}^{l} e_{ik}$, $l \geq 1$ then there exists a path from p_1 to p_2 given by a sequence of adjacent points in the grid and that this sequence is determined by a permutation of $e_{ik}s$.
Inductive step: Let us consider the case $s_i = \sum_{k=1}^{l+1} e_{ik}$, $l \geq 1$.
case 1: All the $c_{ik2}s$ are positive. Since p_2 is in the grid, $\sum_{k=1}^{l+1} c_{ik2} \leq width$ This implies that there exists a point p_3 in the grid and an e_{ij} such that $P_3 + e_{ij} = P_2$. Since vector addtion is commutative and the length of the path from p_1 to p_3 is l, by induction hypothesis there is a path of adjacent points from p_1 to p_2 in the grid.
case 2: All the $c_{ik2}s$ are negative. Proof is very similar to case one.
case 3: There is at least one positive $c_{ik2}s$ and one negative $c_{ik2}s$. Let $P_2 = (cP_{21}, cP_{22})^T$ is the vector representing point p_2. Suppose $cP_{22} \leq p_{max}$, we can find a point p_3 such that $P_3 + e_{ik}$ is equal to P_2 and $e_{ik2} < 0$. The $cP_{32} \leq p_{max} + an_{min}$ because $cP_{22} \leq p_{max}$ and the an_{min} is less than the width of the grid. The point p_3 is in the grid because $cP_{11} > 0$ and $cP_{21} \geq cP_{31}$. From the induction hypothesis there is a path of adjacent points within the grid from p_1 to p_2 and it corresponds to a permutation of $e'_{ik}s$.

When $cP_{22} > p_{max}$ we can find a e_{ik} and a point p_3 such that $P_3 + e_{ik}$ is equal to P_2 and $e_{ik2} > 0$. The $cP_{32} > 1$ is true, because $cP_{22} > p_{max}$. The point p_3 is in the grid because $cP_{11} > 0$ and $cP_{21} \geq cP_{31}$. From the induction hypothesis, there is a path of adjacent points within the grid from p_1 to p_2 and it corresponds to a permutation of $e'_{ik}s$. □

We show that when the width of the rectangular iteration space is greater than or equal to the sum of p_{max} and an_{min}. If an edge is redundant at a point in an S_ISDG then it is redundant at every other point.

Theorem 3. If a dependence edge is redundant at a point of the S_ISDG of a rectangular two dimensional loop and the width of the iteration space is greater than $p_{max} + an_{min}$, then it is redundant at all points.

Proof: Let us assume that the dependence d_i is redundant at point p_1. Let p_2 be an arbitray point. Let P_1 and P_2 be the vectors representing the points p_1 and p_2, respectively.
Case 1: $P_2 + d_i \trianglerighteq top$
This implies that the edge corresponding to the dependence d_i does not exist at point p_2. Hence, the theorem is obviously true.
Case 2: $P_2 + d_i \trianglelefteq bottom$
Because all the components of any d_{ij} are positive, it is not possible to appear. Hence, the theorem is obviously true.
Case 3: $bottom \trianglelefteq P_2 + d_i \trianglelefteq top$
The condition one for the redundancy is vacuously true. Let p_3 be the adjacent point of p_2 by distance d_i. Since the width of the iteration space is greater than or equal to sum of p_{max} and an_{min}, we can always find a rectangular grid, region of width equal to sum of p_{max} and an_{min} enclosing p_2 and p_3. From Lemma 1 there exists an alternate path in the selected region from point p_2 to p_3. Hence, conditions two and three of redundancy are satisfied. Therefore the edge due to dependence d_i is redundant at any point p_2. □

At above, we have given some theorems about how to delete redundant synchronizations from a regular loop. In fact, there are a lot of triangle loops and irregular loops in the scientific calculation. It is can be proved that when all components of dependence vetors are positive, the redundancy can be deleted from all points in triangle loop iteration space. Before Theorem 4, we prove the following lemma.

Lemma 2. If none of the dependence vector of a triangle loop has negtive components, let p_1 be a point in a triangle iteration space and $s_i = \sum_{k=1}^{n} e_{ik}$, where $n \geq 1$ and $e_{ik} \in \Delta$ not necessarily distinct. From any point within a triangle , if the adjacent point due to s_i is within the triangle, then there exists a path of length n given by a sequence of adjacent points in the triangle that corresponds to some permutation of $e'_{ik}s$.

Proof: We prove this using induction on the length of the path, that is the number of $e'_{ik}s$. Let p_2 be the point given by $P_1 + s_i$, and $p_1 = (i_1, j_1)$, $p_2 = (i_2, j_2)$, $p_0 = (i_2, j_1)$, we will prove that the path is in the triangle $\Delta p_0 p_1 p_2$.
Basis: When k is equal to one, s_i is equal to e_{i1}. Since k is equal to one, there is only one permutation and the path is given by p_1, p_2. It is clear that s_i is in the triangle $\Delta p_0 p_1 p_2$.
Hypothesis: Let us assume that for any $s_i = \sum_{k=1}^{l} e_{ik}$, $l \geq 1$ then there exists a path from p_1 to p_2 given by a sequence of adjacent points in the triangle $\Delta p_0 p_1 p_2$ and this sequence is determined by a permutation of $e_{ik}s$.
Inductive step: Let us consider the case $s_i = \sum_{k=1}^{l+1} e_{ik}$, $l \geq 1$, here s_i is denoted by its components $(C_{i1}, C_{i2})^T$. Because $\sum_{i=1}^{l+1} C_{i1}$ is equal to i_2, $\sum_{i=1}^{l+1} C_{i2}$ is equal to j_2, and $P_1 + s_i = p_2$. We can find a point $p_3 = (i_3, j_3)$, which is in the triangle $\Delta p_0 p_1 p_2$, and an e_{ij} such that $P_3 + e_{ij} = P_2$.Since vector addtion is commutative and the length of the path from p_1 to p_3 is l, by induction hypothesis there is a path of adjacent points from p_1 to p_2 in $\Delta p_0 p_1 p_2$. □

Theorem 4. If none of the dependence vector of a triangle loop has negtive components, then the redundancy of a dependence vector at any point of the S_ISDG implies its redundancy at all points of the S_ISDG.

Proof: Let us assume that the dependence d_i is redundant at a point. Let p_1 be an arbitray point. Let P_1 be the vetors representing the points p_1. Because $d_i = \sum_{j=1}^{n} d_{ij}$, from Lemma 2 there exists an alternate path in the selected region corresponding to point p_1 and $P_1 + d_i$. The selected region is in the triangle iteration space, therefore, the alternate path of $d_i = \sum_{j=1}^{n} d_{ij}$ is also in the triangle iteration space. □

In some situations, we must apply the local optimization to some areas, because the dependency is not uniform. we can not give a theorem which represents a method of deleting redundant synchronizations from all triangle loops and irregular loops. Now, we give a theorem for triangle loops. It shows a method of analysing redundant synchronizations in triangle loops and irregular loops. The following triangle loop (called $\Delta -$ *triangle loop* for short) is used in representing it.

```
DO I = 1, $N$
  DO J = 1, $a * I + b$
      ...              $a = (1 - M)/(N - 1)$,
      ...              $b = (MN - 1)/(N - 1)$.
  ENDDO
ENDDO
```

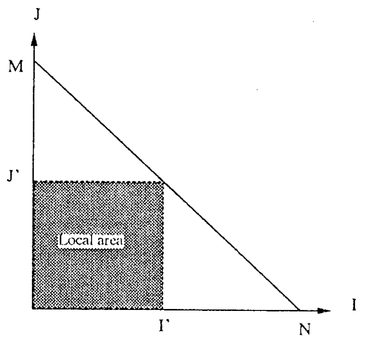

Figure 4: The pattern of $\Delta -$ *triangle loop*

Theorem 5. If a dependence edge is redundant at a point of a $\Delta -$ *triangle loop* and $(1/2)(MN - 1)/(N-1)$ is greater than $p_{max} + an_{min}$, this redundancy can be deleted from $(1/4)(MN - 1)^2/((M - 1)(N - 1))$ points of the S_ISDG at least. The ratio of deleted dependences among all points is $(1/2)$ $(MN - 1)^2$ / $(MN(M - 1)(N - 1))$.

Proof: In a $\Delta -$ *triangle loop* iteration space, a largest rectangular area can be found. Assume this area as the local area of optimization. In Fig.4, the number of iterations in a rectangular is $I' * J'$.

$$
\begin{aligned}
S = I' * J' &= I' * (a * I' + b) \\
&= a * I'^2 + b * I \\
&= ((1 - M)/(N - 1))I'^2 + \\
&\quad ((MN - 1)/(N - 1))I'
\end{aligned}
$$

It is clear that when $I' = -b/(2a)$, the rectangular is the largest,

$$I' = (1/2)(MN - 1)/(M - 1)$$

$$J' = (1/2)(MN - 1)/(N - 1)$$

from theorem 3, when J' is greater than $p_{max} + an_{min}$, *i.e.*, $(1/2)(MN-1)/(N-1) > p_{max}+an_{min}$, the dependence at all points in this rectangular can be deleted. Therefore,the number of dependences which can be delete at least from S_ISDG is $(1/4)$ $(MN - 1)^2/((M - 1)(N - 1))$.

The total number of points in a $\Delta -$ *triangle loop* is $(MN)/2$, the ratio of deleted dependences is $2I'J'/(MN)$, *i.e.*, $(1/2)(MN - 1)^2/(MN(M - 1)(N - 1))$. □

Theorem 5 means that we can find a largeset rectangular and regard it as a local area for the local optimization in a triangle loop or a irregular loop. In a $\Delta -$ *triangle loop* iteration space, when M and N are much larger than one , a redundant syncronization can be deleted from more than a half of number of points in the S_ISDG.

According to the theorems, we delete redundant edges from n-nested loops. Let Δ be the set of dependence vector. $\Delta = \{ d_1, d_2, ..., d_m \}$. Let d_j be any dependence vector. $d_j = (i_{j1}, i_{j2}, ..., i_{jn})$ Let max_l be the maximum of the number l subscript of d_j. $max_l = \sum_{l=1}^{m} i_{jl}$, $i_{jl} \geq 0$. At first, we construct a subgraph *Subg(bottom)* rooted at *bottom*. The bound of *Subg(bottom)* are from 1 to max_l, $1 \leq l \leq n$. According to the relation of dependence, dependence edges will be represented in the *Subg(bottom)*. Let node 1 be $bottom = (1, 1, ..., 1)$, we apply the V_DFS algorithm at node 1, a spanning tree corresponding to the subgraph of S_ISDG will be created. Due to the node 1, deleted edges are redundant edges. Dependences corresponding to these deleted edges can be deleted from every points in the iteration space or local area of the iteration space.

3 Conclusions

Previous work considers only simple loops, whereas, we have investigated and characterized redundant synchronizations in n-nested loops. This algorithm can be implemented as a phase in parallel compilers. The reduction in parallel execution time is dependent on the type of synchronization primitives avaliabale and the scheduling scheme.

Some theorems are presented for deleting redundant synchronizations from n-nested loops. The methods of deleting them from triangle loops also have been shown. Accordiŋg to theorems, the scheme using S_ISDG is proposed. One of the technique proposed for execution of loops on a distributed memory computer is iteration space tiling. The tile size is dependent on the dependences. Elimination of redundant dependences in the S_ISDG can potentiallly reduce the tile size and increase the concurrency. This scheme is also applicable for presynchronized scheduling on share memory multiprocessors. We have also verified the correctness of this algorithm with 30 loops in the simulating enviorment.

References

[1] Cytron,R.G.: *Compiler-time Scheduling and Optimization for Asynchronous Machines*,Ph.D. dissertation, Univ. Illinois, Urbana-Champaign, DCS Rep. UIUCDCS-R-84-1177, Oct. 1984.

[2] Padua,D.A., Kuck,D.J.,and Lawrie,D.H.: *High-speed Multiprocessors and Compilition Techniques*, *IEEE Tran. Comput*, Vol.29, No.9(1980), pp.763-776.

[3] Midkiff,S.P., and Padua,D.A.: *Compiler Algorithms for Synchronization*, *IEEE Tran. Comput*, Vol.36, No.12(1987), pp.1485-1495.

[4] Lee,S.C., and Harada,K.:*An Algorithm for Deleting Redundant Synchronizations from Parallel Loops*, *9th Conference Proceedings Japan Society for Software Science and Technology*, pp.489-492, 1992.

[5] Wolfe,M.J.: *Optimizing Supercompilers for Supercomputers*, The MIT Press, Cambridge, 1988.

[6] Tarjan,R.: *Depth First Search and Linear Graph Algorithms*, *SIAM J. Comput*, No.1(1972), pp.146-160.