JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 17, 463-489 (2001)

An Optimized Three Region Partitioning Technique to Maximize Parallelism of Nested Loops with Non-uniform Dependences

DER-LIN PEAN AND CHENG CHEN
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.
E-mail: dlpean@csie.nctu.edu.tw

There are many methods for nested loop partitioning exist; however, most of them perform poorly when they partition loops with non-uniform dependences. This paper proposes a generalized and optimized loop partitioning mechanism which can exploit parallelism in nested loops with non-uniform dependences. Our approach based on the region partitioning technique divides the loop into variable size partitions. Furthermore, the proposed algorithm partitions a nested loop using the copy-renaming and optimized partitioning techniques so as to minimize the serial part of the iteration space. Thus, it out performs previous partition mechanisms for nested loops with non-uniform dependences. Compared with other popular techniques, our scheme shows dramatic improvement in preliminary performance results.

Keywords: compilers, non-uniform dependence, loop parallelization, parallel compiler, parallel processing, region partitioning technique

1. INTRODUCTION

The dependence of loops can be divided into two categories: one is uniform dependences and the other, non-uniform dependences [8]. When a pattern of dependence vectors, namely, distance vectors, is expressed by some constants, it will be called uniform dependence. While other dependence vectors in a regular pattern can not be expressed by constants, they belong to non-uniform dependences. Example 1 below explicates a non-uniform dependence loop, which has non-uniform dependence in the iteration space (See Fig. 1).

Example 1: A non-uniform dependence loop:

```
for   I = 1, 10
  for   J = 1, 10
    A (2*I + 3, J + 1) = …..
                  ….. = A (2*J + I + 1, I + J + 3)
  endfor
endfor
```

Received May 19, 1999; revised July 21, 1999; accepted November 15, 1999.
Communicated by Jang-Ping Sheu.

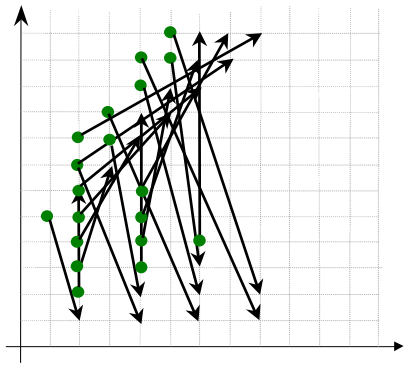


Fig. 1. Iteration space of Example 1.

Because rich parallelism in loops exists in scientific programs, current parallelizing compilers have been developed in an effort to exploit this parallelism [8]. However, most of them fail to parallize nested loops with non-uniform dependences because of irregular and complex dependence constraints. Although many studies have tried to identify the cross-iteration dependences in nested loops, most of them have failed when analysis was conducted using coupled subscripts [9].

According to an empirical study on array subscripts and data dependences [1], nearly 45 % of two-dimensional array references are coupled, and most of them generate non-uniform dependences. In this paper, we will focus on the parallelization of nested loops with non-uniform dependences. Loop partitions are an important optimization issue requiring exact and effective data dependence analysis [3]. However, irregularity in a dependence pattern makes dependence analysis of nested loops very difficult. A number of techniques based on convex hull theory have been proposed, such as dependence uniformization [2, 11, 16], minimum dependence distance tiling [4, 10, 12, 14, 15], three region partitioning [5], unique set oriented partitioning [6], and ITRP (Improved Three Region Partitioning) [7]. However, none of them can extract all of the parallelism from non-uniform dependence loops.

We employ a mechanism called OTRP (Optimized Three Region Partitioning), which divides the iteration space into two parallel regions and one serial region. The serial region can be further parallelized into parallel tiles using general non-uniform parallelization techniques, such as minimum dependence distance tiling [4, 10, 12, 14, 15], or variable size partitioning techniques [7]. In our preliminary performance evaluation, this method performed 1.25 to 1.5 times better than the currently most effective method, called ITRP, when we ran a simulation on the system with 8 processors. On the other hand, in order to fully determine the parallelism obtained by our method, we ran our scheme on a simulation environment with up to 128 processors. We also found that our method performed 1.1 to 1.3 times better than the ITRP method. When we increased the loop bound of the benchmark from 10 to 100 and 1000 to simulate the property of real scientific benchmarks, our technique always performed better than the currently popular mechanisms.

The rest of this paper is organized as follows. Section 2 describes our program model and reviews the concepts of the dependence convex hull, unique head and tail sets

as will as several related works. Section 3 presents the concept and principle behind our new partitioning mechanism, the OTRP scheme. Section 4 compares our scheme with previously proposed schemes. The preliminary performance results are presented in section 5. Finally, we give some conclusions along with future work in section 6.

2. PRELIMINARIES AND RELATED WORK

Most loops with complex array subscripts are two dimensional loops [1]. To simplify, explanation, the program under consideration in this paper is a doubly nested loop with coupled subscripts. Solutions for multilevel nested loops can be obtained by enhancing our mechanisms. The model of a doubly nested loop is depicted in Fig. 2, where $f_1(I, J)$, $f_2(I, J)$, $f_3(I, J)$, and $f_4(I, J)$ are all linear functions of loop variables. The dimension of the nested loop is equal to the number of nested loops in it. In loop I(J), $L_I(L_J)$ and $U_I(U_J)$ indicate the lower and upper bounds, respectively. Both the lower and upper bounds for indices should be known at compiler time.

for I = L_I, U_I
  for J = L_J, U_J
    ……
    S_d: A($f_1(I, J)$, $f_2(I, J)$) = ….
    S_u: …... = A($f_3(I, J)$, $f_4(I, J)$)
    ……
  endfor
endfor

Fig. 2. A doubly nested loop program model.

Usually, an iteration denoting a series of statements in the loop body is a unit of work assigned to a processor. Therefore, the dependence constraints inside the iteration can be ignored when parallelizing a nested loop. The dependence constraints among different iterations, called cross-iteration dependences, are our major concern. In our program model shown in Fig. 2, statement S_d defines the elements of array A, and statement S_u uses them. Dependence exists between S_d and S_u whenever both refer to the same element in array A. If the element defined by S_d is used by S_u in a subsequent iteration, a cross-iteration flow dependence exists between S_d and S_u and will be denoted by $S_d \ \delta^f \ S_u$. On the other hand, if the element used in S_u is defined by S_d at a later iteration, the dependence is called a cross-iteration anti-dependence and will be denoted also by $S_u \ \delta^a \ S_d$.

The most common method for computing data dependences involves solving a set of linear Diophantine Eqs. with a set of constraints formed by the iteration boundaries. The loop shown in Fig. 2 carries cross-iteration dependences if and only if there exist four integers i_1, j_1, i_2 and j_2 satisfying the system of linear Diophantine Eqs. given by (1) and the system of inequalities given by (2). The formal expression of Eq. (1) is shown in Eq. (1-1):

$$f_1(i_1, j_1) = f_3(i_2, j_2) \text{ and } f_2(i_1, j_1) = f_4(i_2, j_2), ((i_1, j_1) \text{ and } (i_2, j_2) \in (I, J)), \quad (1)$$

$$a_{11}i_1 + a_{12}j_1 + b_1 = a_{31}i_2 + a_{32}j_2 + b_3 \text{ and } a_{21}i_1 + a_{22}j_1 + b_2 = a_{41}i_2 + a_{42}j_2 + b_4, ((i_1, j_1) \text{ and } (i_2, j_2) \in (I, J)), \quad (1\text{-}1)$$

$$L_1 \leqq i_1, i_2 \leqq U_1 \text{ and } L_2 \leqq j_1, j_2 \leqq U_2 . \quad (2)$$

The Dependence Convex Hull (DCH) [2] is a convex polyhedron and a subspace of the solution space. There are two ways to solve the system of Diophantine Eqs. in (1). One way is to set i_1 to x_1, set j_1 to y_1, and solve i_2 and j_2. Then, i_1, j_1, i_2, j_2 and its inequalities can be represented as (3), which forms DCH, denoted DCH1. The formal expression of Eq. (3) is shown in Eq. (3-1):

$$(i_1, j_1, i_2, j_2) = (x_1, y_1, g_1(x_1, y_1), g_2(x_1, y_1)),$$

$$L_1 \leqq x_1, g_1(x_1, y_1) \leqq U_1 \text{ and } L_2 \leqq y_1, g_2(x_1, y_1) \leqq U_2 , \quad (3)$$

$$g_1(x_1, y_1) = ((a_{42}a_{11} - a_{32}a_{21})x_1 + (a_{42}a_{12} - a_{32}a_{22})y_1 + (a_{42}b_1 - a_{32}b_2 - a_{42}b_3 + a_{32}b_4)) / (a_{42}a_{31} - a_{32}a_{41}),$$

$$g_2(x_1, y_1) = ((a_{41}a_{11} - a_{31}a_{21})x_1 + (a_{41}a_{12} - a_{31}a_{22})y_1 + (a_{41}b_1 - a_{31}b_2 - a_{41}b_3 + a_{31}b_4)) / (a_{41}a_{32} - a_{31}a_{42}),$$

$$L_1 \leqq x_1, g_1(x_1, y_1) \leqq U_1 \text{ and } L_2 \leqq y_1, g_2(x_1, y_1) \leqq U_2 . \quad (3\text{-}1)$$

The other way is to set i_2 to x_2, set j_2 to y_2 and solve i_1 and j_1. Then, (i_1, j_1, i_2, j_2) and its inequalities can be represented as in (4), which forms a DCH denoted by DCH2. The formal expression of Eq. (4) is shown in Eq. (4-1):

$$(i_1, j_1, i_2, j_2) = (g_3(x_2, y_2), g_4(x_2, y_2), x_2, y_2),$$

$$L_1 \leqq g_3(x_2, y_2), x_2 \leqq U_1 \text{ and } L_2 \leqq g_4(x_2, y_2), y_2 \leqq U_2 , \quad (4)$$

$$g_3(x_2, y_2) = ((a_{22}a_{31} - a_{12}a_{41})x_2 + (a_{22}a_{32} - a_{12}a_{42})y_2 + (a_{22}b_3 - a_{12}b_4 - a_{22}b_1 + a_{12}b_2)) / (a_{22}a_{11} - a_{12}a_{21}),$$

$$g_4(x_2, y_2) = ((a_{21}a_{31} - a_{11}a_{41})x_2 + (a_{21}a_{32} - a_{11}a_{42})y_2 + (a_{11}b_3 - a_{11}b_4 - a_{21}b_1 + a_{11}b_2)) / (a_{21}a_{12} - a_{11}a_{22}),$$

$$L_1 \leqq g_3(x_2, y_2), x_2 \leqq U_1 \text{ and } L_2 \leqq g_4(x_2, y_2), y_2 \leqq U_2 . \quad (4\text{-}1)$$

Clearly, if we have a solution i_1, j_1 in DCH1, then we will have a solution i_2, j_2 in DCH2 because both of them are derived from the same set of Eqs.. The union of DCH1 and DCH2 is called the Complete DCH (CDCH), and all the dependences lie within CDCH. If iteration (i_2, j_2) is dependent on iteration (i_1, j_1), then we will have a dependence vector D(x, y) with $d_i(x, y) = i_2 - i_1$ and $d_j(x, y) = j_2 - j_1$. Therefore, for DCH1, we

have

$$d_i(i_1, j_1) = g_1(i_1, j_1) - i_1 \text{ and } d_j(i_1, j_1) = g_2(i_1, j_1) - j_1. \quad (5)$$

The formal expression of Eq. (5) is shown in Eq. (5-1):

$$\begin{aligned} d_i(i_1, j_1) &= ((a_{42}a_{11} - a_{32}a_{21} - a_{42}a_{31} + a_{32}a_{41})i_1 + (a_{42}a_{12} - a_{32}a_{22})j_1 + \\ &\quad (a_{42}b_1 - a_{32}b_2 - a_{42}b_3 + a_{32}b_4)) / (a_{42}a_{31} - a_{32}a_{41}), \\ d_j(i_1, j_1) &= ((a_{41}a_{11} - a_{31}a_{21})i_1 + (a_{41}a_{12} - a_{31}a_{22} - a_{41}a_{32} + a_{31}a_{42})j_1 + \\ &\quad (a_{41}b_1 - a_{31}b_2 - a_{41}b_3 + a_{31}b_4)) / (a_{41}a_{32} - a_{31}a_{42}). \end{aligned} \quad (5\text{-}1)$$

For DCH2, we have

$$d_i(i_2, j_2) = i_2 - g_3(i_2, j_2) \text{ and } d_j(i_2, j_2) = j_2 - g_4(i_2, j_2). \quad (6)$$

The formal expression of Eq. (6) is shown in Eq. (6-1):

$$\begin{aligned} d_i(i_2, j_2) &= ((a_{22}a_{11} - a_{12}a_{21} - a_{22}a_{31} - a_{12}a_{41})i_2 - (a_{22}a_{32} - a_{12}a_{42})j_2 + \\ &\quad (a_{22}b_3 - a_{12}b_4 - a_{22}b_1 + a_{12}b_2)) / (a_{22}a_{11} - a_{12}a_{21}), \\ d_j(i_2, j_2) &= ((a_{11}a_{41} - a_{21}a_{31})i_2 + (a_{21}a_{12} - a_{11}a_{22} - a_{12}a_{32} + a_{11}a_{42})j_2 + \\ &\quad (a_{11}b_4 + a_{21}b_1 - a_{21}b_3 - a_{11}b_2)) / (a_{21}a_{12} - a_{11}a_{22}). \end{aligned} \quad (6\text{-}1)$$

Here, we briefly describe some techniques for solving non-uniform dependence problems. The dependence uniformization scheme [2] constructs two basic dependence vector sets based on the dependence slope theory and adds them to every iteration in the iteration space. The loop can then be parallelized current parallel compilation techniques for uniform dependence loops. It can be parallelized according to the two uniform dependence vectors, resulting in a doacross type of loop execution. However, this mechanism always imposes many additional dependences on the iteration space.

The minimum dependence distance tiling method [4, 15] exploits the available parallelism using minimum distances computed from the dependence vectors of the IDCH (Integer DCH) extreme points. The minimum distances are used to partition the iteration space into tiles of regular size and shape, but the irregularity of non-uniform dependence distances is ignored.

The three-region partitioning technique [5] divides the iteration space into two parallel regions and one serial region. The first region represents the part of the iteration space in which anti-dependence exists. Hence, iterations in this area can be fully executed in parallel provided that copy-renaming is performed. The second region represents the part of the iteration space having flow dependence heads whose corresponding tails exist in the first region. Iterations in the second region are executed in parallel after the those in the first region are executed. Then, the serial region represents the rest of the iteration space to which the dependence uniformization scheme can be applied. However, if this serial region increases, the performance of the loop will be significantly degraded.

The unique set oriented partitioning mechanism [6] divides the iteration space into the dependence unique head and tail sets. In this partitioning method, there exist various combinations of overlaps of these sets, and the execution order of the partitioned regions depends on these combinations. The iterations within the unique set can be executed in parallel, but the unique tail sets must be executed before the unique head sets. This technique provides more accurate information for the iteration space. However, it also has several disadvantages. First, it does not present an exact partitioning scheme, so it is difficult to know which is the best scheme at compile time. Second, it inevitably leaves some parallelism unexplored since the method of minimum dependence distance is applied to parallelize the rest of the iteration space which contains both dependence tails and heads. Increasing the number of iterations in this region will reduce the speedup of the loop.

The Improved Three Region Partition (ITRP) scheme [7] is similar to the three region partitioning scheme in the sense that the iteration space is divided into two parallel regions and one serial region. The size of the parallel region in ITRP is not less than that in the three-region partitioning scheme mentioned above. On the other hand, it is simple to divide the iteration space into three regions. The execution order of partitions will always be the same. However, this still leaves some parallelism unexplored in the partitioned serial region. Moreover, variable size partitioning in the serial region also suffers from high time complexity.

3. OPTIMIZED THREE REGION PARTITIONING

In the previous section, we briefly discussed the advantages and disadvantages of conventional techniques. Here, we will present an effective technique, called OTRP (Optimized Three Region Partitioning), which can improve the drawbacks of those methods.

We use an arrow to represent dependences in the iteration space. We call the arrow's Head the dependence head and the arrow's Tail the dependence tail. Definition 1 defines the unique head (tail) set. If any region belongs to the unique set, the iterations in it can be executed in parallel [6].

Definition 1[6]: (Unique Head (Tail) Set) The unique head (tail) set is a set of integer points in the iteration space that satisfies the following conditions:

(1)It is the subset of one of the DCHs (or is the DCH itself).
(2)It contains all the dependence arrows' heads (tails) but does not contain any other dependence arrows' tails (heads).

We will first examine the concepts of DCH1 and DCH2.

Lemma 1: DCH1 contains all flow dependence tails and all anti-dependence heads (if they exist), and DCH2 contains all anti-dependence tails and all flow dependence heads (if they exist).

Proof: From Eqs. (3) and (4), the iterations in DCH1 and DCH2 can be determined. In addition, from Eqs. (5) and (6), the dependence vectors of DCH1 and DCH2 can also be calculated. The iterations inside DCH1 can be constructed by means of the functions $g_1(i_1, j_1)$ and $g_2(i_1, j_1)$. Thus, if there is any iteration inside DCH1, and if its corresponding array reference is a write reference, then the array reference of the iteration must become a flow dependence tail. Otherwise, if there is any iteration inside DCH1, and if its corresponding array reference is a read reference, then the array reference of the iteration must become an anti-dependence head. Hence, DCH1 contains all flow dependence tails and all anti-dependence heads (if they exist). Similarly, DCH2 contains all anti-dependence tails and flow dependence heads (if they exist).

The above lemma tells us that DCH1 and DCH2 may contain more than one unique set. Two kinds of unique sets in DCH1 and DCH2 are also given by Lemma 1. We can use this property to further partition the iteration space inside DCH1 and DCH2 into sub-iteration spaces to be executed in parallel. The following lemma states the condition for DCH1 and DCH2 to be unique sets.

Lemma 2: If $d_i(x, y) = 0$ does not pass through any DCH, then there will be only one kind of dependence, either flow or anti-dependence, and DCH itself will be the unique head set or the unique tail set.

Proof: If $d_i(x, y) = 0$ does not pass through any DCH, then DCH is on the side of either $d_i(x, y) < 0$ or $d_i(x, y) > 0$. If DCH1 and DCH2 are on the side of $d_i(x, y) > 0$, then DCH1 and DCH2 contain a flow dependence unique tail set and a flow dependence unique head set, respectively, because the iterations in DCH1 and DCH2 are derived from Eqs. (5) and (6), respectively, and are based on Lemma 1. If DCH1 and DCH2 are on the side of $d_i(x, y) < 0$, then DCH1 and DCH2 contain an anti-dependence unique head set and an anti-dependence unique tail set, respectively, because the iterations in DCH1 and DCH2 are derived from Eqs. (5) and (6), respectively, and are based on Lemma 1. Thus, if $d_i(x, y) = 0$ does not pass through any DCH, then there will be only one kind of dependence, either flow or anti-dependence, and DCH itself will be the unique head set or the unique tail set.

DCH1 and DCH2 are constructed based on the same system of linear Diophantine Eqs. and inequalities. Lemma 3 highlights their common attributes.

Lemma 3: If $d_i(x_1, y_1) = 0$ does not pass through DCH1, then $d_i(x_2, y_2) = 0$ will not pass through DCH2.

Proof: If $d_i(x_1, y_1) = 0$ does not pass through DCH1, then there is only one kind of dependence, either flow or anti-dependence, according to Lemma 2. Now, assume that $d_i(x_2, y_2) = 0$ passes through DCH2; then, DCH2 contains both a flow dependence head set and an anti-dependence tail set. Thus, there must be a corresponding flow dependence tail set and an anti-dependence head set inside DCH1 due to Lemma 1. However, this means that $d_i(x_1, y_1) = 0$ does pass through DCH1 according to Lemma 2. Henceforth, if $d_i(x_1, y_1) = 0$ does not pass through DCH1, then $d_i(x_2, y_2) = 0$ will not pass

through DCH2.

Lemma 4: If $d_i(x_1, y_1) = 0$ ($d_i(x_2, y_2) = 0$) does not pass through DCH1(DCH2), and if DCH1(DCH2) is on the side of $d_i(x_1, y_1) > 0$ ($d_i(x_2, y_2) > 0$), then DCH1(DCH2) is a flow dependence unique tail (head) set. Otherwise, if DCH1 (DCH2) is on the side of $d_i(x_1, y_1) < 0$ ($d_i(x_2, y_2) < 0$), then DCH1(DCH2) is an anti-dependence unique head (tail) set.

Proof: If $d_i(x_1, y_1) = 0$ ($d_i(x_2, y_2) = 0$) does not pass through DCH1 (DCH2), and if DCH1 (DCH2) is on the side of $d_i(x_1, y_1) > 0$ ($d_i(x_2, y_2) > 0$), then DCH1 will be a flow dependence unique tail set according to Lemma 1 and Lemma 2. On the other hand, DCH2 will be a flow dependence unique head set due to Lemma 3. Otherwise, if DCH1 (DCH2) is on the side of $d_i(x_1, y_1) < 0$ ($d_i(x_2, y_2) < 0$), then DCH1 will be an anti-dependence unique head set based on Lemma 1 and Lemma 2. On the other hand, DCH2 will be an anti-dependence unique tail set due to Lemma 3.

We have now established that if $d_i(x_1, y_1) = 0$ does not pass through DCH1, then both DCH1 and DCH2 are unique sets and the points in them have the same property. When $d_i(x_1, y_1) = 0$ ($d_i(x_2, y_2) = 0$) passes through DCH1(DCH2), DCH1(DCH2) may contain both dependence heads and tails. This makes it harder to find the unique head and tail sets. We now will propose an effective mechanism for exploiting parallelism under such conditions. Lemmas 5 and 6 show some common attributes when $d_i(x_1, y_1) = 0$ passes through DCH1(DCH2).

Lemma 5: If $d_i(x, y) = 0$ passes through a DCH, then it will divide the DCH into a unique tail set and a unique head set. Furthermore, $d_j(x, y) = 0$ will determine the inclusion of $d_i(x, y) = 0$ in one of the sets.

Proof: If $d_i(x, y) = 0$ passes through DCH1(DCH2), then DCH1(DCH2) will contain both a flow dependence tail(head) set and an anti-dependence head(tail) set, and the sets will be divided by the line $d_i(x, y) = 0$, according to Eqs. (3), (4), (5), and (6). The points in the line $d_i(x, y) = 0$ can be further categorized into different sets according to the line $d_j(x, y) = 0$. If these points are on the side of $d_j(x, y) > 0$, then they belong to a flow dependence unique tail set (flow dependence unique head set) in DCH1(DCH2). Otherwise, if these points are on the side of $d_j(x, y) < 0$, then they belong to an anti-dependence unique head set (anti-dependence unique tail set) in DCH1(DCH2) according to the calculation results of Eqs. (5) and (6). Furthermore, if the point is in the $d_i(x, y) = 0$ and $d_j(x, y) = 0$, it will have no cross-iteration dependence. Thus, $d_j(x, y) = 0$ will determine the inclusion of $d_i(x,y) = 0$ in one of the sets.

Lemma 6: If $d_i(x_1, y_1) = 0$ passes through DCH1(DCH2), then DCH1(DCH2) is the union of a flow dependence unique tail(head) set and an anti-dependence unique head (tail) set.

Proof: If $d_i(x, y) = 0$ passes through DCH1(DCH2), then DCH1(DCH2) contains both a flow dependence unique tail(head) set and an anti-dependence unique head(tail) set, and the sets are divided by the line $d_i(x, y) = 0$, according to Lemmas 1 and 5. Thus, if $d_i(x_1,$

y_1) = 0 passes through DCH1(DCH2), then DCH1(DCH2) is the union of a flow dependence unique tail(head) set and an anti-dependence unique head (tail) set.

Based on the properties described above, there exist various combinations of overlaps of these unique sets. We will explain these properties using the following example:

Example 2: An example of a doubly nested loop:

```
for I = 1, 10
  for J = 1, 10
    A (2*J + 3, I + J + 5) = …..
                        ….. = A (2*I + J - 1, 3*I - 1)
  endfor
endfor
```

The set of inequalities and dependence distances of the loop in Example 2 is computed as follows:

DCH1: (7)

$$1 \le i_1 \le 10 \quad and$$
$$1 \le j_1 \le 10 \; and$$
$$1 \le \frac{i_1}{3} + \frac{j_1}{3} + 2 \le 10 \; and$$
$$1 \le -\frac{2i_1}{3} + \frac{4j_1}{3} \le 10$$
$$d_i(i_1, j_1) = -\frac{2i_1}{3} + \frac{j_1}{3} + 2,$$
$$d_j(i_1, j_1) = -\frac{2i_1}{3} + \frac{j_1}{3} .$$

DCH2: (8)

$$1 \le i_2 \le 10 \quad and$$
$$1 \le j_2 \le 10 \; and$$
$$1 \le 2i_2 - \frac{j_2}{2} - 4 \le 10 \; and$$
$$1 \le i_2 + \frac{j_2}{2} - 2 \le 10$$
$$d_i(i_2, j_2) = -i_2 + \frac{j_2}{2} + 4,$$
$$d_j(i_2, j_2) = -i_2 + \frac{j_2}{2} + 2.$$

Fig. 3 DCHs and the unique head (tail) sets of the loop in Example 1. Clearly,

$d_i(x_1, y_1) = 0$ divides DCH1 into two areas. The area on the side of $d_i(x_1, y_1) < 0$ is the anti-dependence unique head set, and the set is on the right side of $d_i(x_1, y_1) = 0$ as shown in Fig. 3(b). The area on the side of $d_i(x_1, y_1) > 0$ is the flow dependence unique tail set, and the set is on the left side of $d_i(x_1, y_1) = 0$ as shown in Fig. 3(a). $d_i(x_2, y_2) = 0$ also divides DCH2 into two areas. The area on the side of $d_i(x_2, y_2) < 0$ is the anti-dependence unique tail set, and the set is on the left side of $d_i(x_2, y_2) = 0$ as shown in Fig. 3(b). The area on the side of $d_i(x_2, y_2) > 0$ is the flow dependence unique head set, and the set is on the right side of $d_i(x_2, y_2) = 0$ as shown in Fig. 3(a).

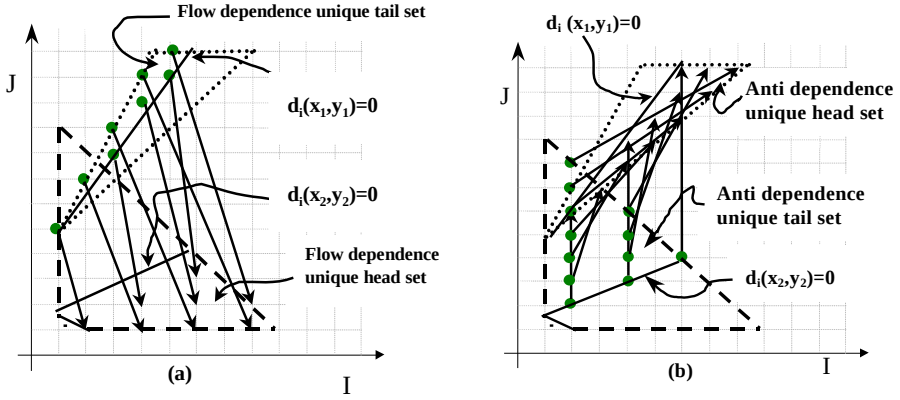


Fig. 3. Unique head sets and unique tail sets of (a) flow dependence and (b) anti-dependence.

Our approach is based on the convex hull theory [2]. We use lines $d_i(i, j) = 0$ and $d_j(i, j) = 0$ to partition the iteration space into two parallel regions and one serial region. All the various kinds of possible sets partitioned by $d_i(i, j)$ and $d_j(i, j)$ are summarized in Table 1 according to the above lemmas. Table 1(b) shows various sets, each of which is part of the line segment corresponding to partitioning according to the sign of $d_j(i_1, j_1)$ and $d_j(i_2, j_2)$.

Table 1. (a) The different sets partitioned by $d_i(i, j)$ and $d_j(i, j)$. (b) The case of $d_i(i_1, j_1) = 0$ or $d_i(i_2, j_2) = 0$.

(a)

		$d_i(i_1, j_1)$		
		> 0	< 0	= 0
$d_i(i_2, j_2)$	> 0	FT, FH DCH_1,DCH_2	FH , AH DCH_2,DCH_1	Refer to (b)
	< 0	FT , AT DCH_1,DCH_2	AT , AH DCH_2,DCH_1	
	= 0	Refer to (b)		

(b)

$d_j(i_1, j_1)$			$d_j(i_2, j_2)$		
> 0	< 0	= 0	> 0	< 0	= 0
FT DCH_1	AH DCH_1	No cross-iteration dependence	FH DCH_2	AT DCH_2	No cross-iteration dependence

FT : Flow dependence Tail set
FH : Flow dependence Head set
AT : Anti dependence Tail set
AH : Anti dependence Head set
DCH_1 and DCH_2 are regions they belong to

First, we use memory space to exploit the benefits of parallel execution because anti-dependence can be avoided based on the concept of copy-renaming [3]. Lemma 7

introduces the condition under which copy-renaming can be used.

Lemma 7: If there exists an anti-dependence between two statements S_d and S_u, denoted as $S_d \ \delta^a \ S_u$, in the iteration space, then these two statements can be executed in parallel after copy-renaming is performed.

Proof: If $S_d \ \delta^a \ S_u$, then we can assume that there are two accesses $A_d \in S_d$ and $A_u \in S_u$ referencing the same memory location. By means of copy-renaming, we copy A_d into another memory location A'_d the iteration is executed. Therefore, access A_u in statement S_u can be changed to A'_u. Thus, the two accesses $A_d \in S_d$ and $A'_u \in S_u$ will not access the same memory location, and the two statements S_d and S_u can be executed in parallel.

Our proposed mechanism, called OTRP (Optimized Three Region Partitioning), can divide the iteration spaces into three regions: $Area_1$, $Area_2$, and $Area_3$. In the following, we explain $Area_1$, $Area_2$, and $Area_3$:

1) $Area_1$: This region may include anti-dependence heads and flow dependence tails, but should not include flow dependence heads. As shown in Table 1, the case of ($d_i(i_2, j_2) < 0$) and ($d_i(i_2, j_2) = 0$ and $d_j(i_2, j_2) < 0$) are included. The case of ($d_i(i_2, j_2) = 0$ and $d_j(i_2, j_2) < 0$) is included because the case ($d_i(i_2, j_2) < 0$) is included. On the other hand, the area in the case of ($d_i(i_1, j_1) > 0$ and $d_i(i_2, j_2) > 0$) subtracting DCH2 is also included. The definition of $Area_1$ is given as follows:

$$Area_1 = Area_{11} \cup Area_{12} \cup Area_{13} \tag{9}$$

where $Area_{11} = \{(i_2, j_2) \mid d_i(i_2, j_2) < 0\}$,

$Area_{12} = \{(i_2, j_2) \mid d_i(i_2, j_2) = 0 \text{ and } d_j(i_2, j_2) < 0\}$,

$Area_{13} = \{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} -$
$\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH2\}$.

Because there may be anti-dependence heads and tails in different areas, copy-renaming is applied to remove anti-dependences based on Lemma 7. Then, the iterations in $Area_1$ can be fully executed in parallel as shown in Theorem 1.

Theorem 1: If a region $Area_1$ contains the following set: $Area_{11} = \{(i_2, j_2) \mid d_i(i_2, j_2) < 0\}$, $Area_{12} = \{(i_2, j_2) \mid d_i(i_2, j_2) = 0 \text{ and } d_j(i_2, j_2) < 0\}$, and $Area_{13} = \{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} - \{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH2\}$, then the iterations in it are a unique flow dependence tail set and can be executed in parallel after copy-renaming is performed.

Proof: $Area_{11}$ contains a flow dependence tail set, an anti-dependence head set and an anti-dependence tail set as shown in Table 1. The anti-dependence head and tail set can be resolved by means of copy-renaming as proved in Lemma 7. $Area_{12}$ contains an

anti-dependence tail set as shown in Table 1 and can also be resolved by means of Lemma 7. The set $\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\}$ contains both flow dependence head and tail sets. The set $\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap \{DCH2\}\}$ contains all the flow dependence head sets in the set $\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\}$ because all the flow dependence sets must exist in DCH2 as presented in Lemma 1. Thus, $Area_{13}$ will contain only flow dependence tail sets. In conclusion, $Area_1$ will contain only the flow dependence tail sets and can be executed in parallel.

2) $Area_2$: In this region, the flow dependence and anti-dependence heads should be included. As shown in Table 1, the case of ($d_i(i_1, j_1) < 0$ and $d_i(i_2, j_2) > 0$), ($d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) < 0$) and ($d_i(i_2, j_2) = 0$ and $d_j(i_2, j_2) > 0$) is included. The case of ($d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) < 0$) and ($d_i(i_2, j_2) = 0$ and $d_j(i_2, j_2) > 0$) is included because the case of ($d_i(i_1, j_1) < 0$ and $d_i(i_2, j_2) > 0$) is included. The definition of $Area_2$ is given by the following set:

$$Area_2 = Area_{21} \cup Area_{22} \cup Area_{23} \cup Area_{24} \tag{10}$$

where $Area_{21} = \{(i_1, j_1) \mid d_i(i_1, j_1) < 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\}$,

$Area_{22} = \{(i_1, j_1) \mid d_i(i_1, j_1) = 0 \text{ and } d_j(i_1, j_1) < 0\}$,

$Area_{23} = \{(i_2, j_2) \mid d_i(i_2, j_2) = 0 \text{ and } d_j(i_2, j_2) > 0\}$,

$Area_{24} = \{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH2\} -$
$\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH1 \cap DCH2\}$.

This $Area_2$ contains anti-dependence heads and flow dependence heads whose corresponding tails exist in $Area_1$ as shown in Theorem 2. Once $Area_1$ and $Area_3$ are executed, the iterations in $Area_2$ can be fully executed in parallel.

Theorem 2: If a region $Area_2$ contains the following set: $Area_{21} = \{(i_1, j_1) \mid d_i(i_1, j_1) < 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\}$, $Area_{22} = \{(i_1, j_1) \mid d_i(i_1, j_1) = 0 \text{ and } d_j(i_1, j_1) < 0\}$, $Area_{23} = \{(i_2, j_2) \mid d_i(i_2, j_2) = 0 \ \& \ d_j(i_2, j_2) > 0\}$, and $Area_{24} = \{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH2\} - \{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH1 \cap DCH2\}$, then there only exists a flow dependence unique head set in it, and it can be executed in parallel after copy-renaming is performed.

Proof: $Area_{21}$ contains both flow dependence head and anti-dependence head sets as shown in Table 1. The anti-dependence head set can be resolved by means of copy-renaming as proved in Lemma 7. $Area_{22}$ contains anti-dependence head sets as shown in Table 1, so the dependence can be avoided by Lemma 7. $Area_{23}$ contains flow dependence head sets as shown in Table 1. The set $\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap DCH2\}$ contains all the flow dependence head sets in the set $\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\}$ because all the flow dependence sets must exist in DCH2 as presented in Lemma 1. On the other hand, $\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid$

$d_i(i_2, j_2) > 0\} \cap$ DCH1 $\cap$ DCH2$\}$ contains flow dependence tails in the set $\{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap$ DCH2$\}$ while DCH1 overlaps with DCH2. $Area_{24}$ contains only flow dependence head sets. In conclusion, $Area_2$ contains only flow dependence unique head sets and can be executed in parallel.

3) $Area_3$: This region is the rest of the iteration space excluding $Area_1$ and $Area_2$. In this region, flow dependence heads and tails are included. As shown in Table 1, the case of $((d_i(i_1, j_1) > 0$ and $d_i(i_2, j_2) > 0) \cap$ DCH1 $\cap$ DCH2) and $(d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) > 0)$ is included. The case of $(d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) > 0)$ is included because the case of $((d_i(i_1, j_1) > 0$ and $d_i(i_2, j_2) > 0) \cap$ DCH1 $\cap$ DCH2) is included. The definition of $Area_3$ is given by the following set:

$$Area_3 = Area_{31} \cup Area_{32} \tag{11}$$

where $Area_{31} = \{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap$ DCH1 $\cap$ DCH2,

$Area_{32} = \{(i_1, j_1) \mid d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) > 0\}$.

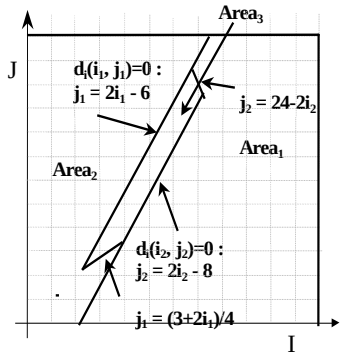
Theorem 3: If a region $Area_3$ contains the following set: $Area_{31} = \{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap$ DCH1 $\cap$ DCH2, and $Area_{32} = \{(i_1, j_1) \mid d_i(i_1, j_1) = 0$ and $d_j(i_1, j_1) > 0\}$, then the iterations in it must be executed in serial.

Proof: $Area_{31} = \{\{(i_1, j_1) \mid d_i(i_1, j_1) > 0\} \cap \{(i_2, j_2) \mid d_i(i_2, j_2) > 0\} \cap$ DCH1 $\cap$ DCH2$\}\}$ contains flow dependence tail and flow dependence head sets while DCH1 overlaps with DCH2. $Area_{32}$ contains only flow dependence tail sets. $Area_3$ contains both flow dependence head and tail sets and must be executed in serial.

Because $Area_1$ contains only flow dependence tail sets after copy-renaming is performed as presented in Theorem 1, and since $Area_2$ contains only flow dependence head sets after copy-renaming is performed as proposed in Theorem 2, they can be executed in parallel, respectively. On the other hand, $Area_3$ contains both flow dependence head and tail sets, so the iterations in it must be executed in a serial manner as shown in Theorem 3. The execution order as follows: first flow dependence tail sets, are executed, followed by the set containing both flow dependence head and tail sets, followed by flow dependence head sets. In other words, partitioning is carried out according to the following sequence $Area_1 \rightarrow Area_3 \rightarrow Area_2$. Iterations in $Area_1$ and $Area_2$ can be executed in parallel, and $Area_3$ must be executed in serial.

$Area_3$ is an inherently serial region. We can apply the minimum dependence distance tiling or dependence uniformization method to parallelize $Area_3$. When our scheme is used, the serial region will be smaller than is the case with other mechanisms. We will compare our scheme with other mechanisms in the next subsection. As shown in Fig. 4(a), the OTRP partitions the iteration space into three regions. Execution of iterations based on three regions can be conducted by transforming the iteration space into three regions (a loop for each region). The upper and lower bounds of the transformed loop can be the lines $d_i(i_2, j_2) = 0$, $d_i(i_1, j_1) = 0$ and the boundary of DCH1 and DCH2, respectively. Fig. 4 illustrates the result of applying the OTRP method to the

loop in Example 2. $Area_1$ is the region satisfying the relation $((J \leqq 2I - 8) \cup ((2I-8 < J \leqq 2I-6) \cap (J > 24-2I)))$, $Area_3$ is $((2I-8 < J \leqq 2I-6) \cap (J > 24-2I) \cap (J > (3 + 2I) / 4))$, and $Area_2$ is $((J > 2I-6) \cup ((2I-8 < J \leqq 2I-6) \cap (J < (3 + 2I) / 4))$. Fig. 4(a) shows the iteration space partitioned into three regions. Fig. 4(b) shows the loops transformed from the loop in Example 2.



(a) Regions of the loop partitioned by the OTRP.

```
/*Area1: region which can be executed in parallel */
Copy A into A'
DOALL I = l1, u1
    DOALL J = l2, min(2I-6, u2 )
      IF((J < 2I-8) | ( J > 24-2I))
      A(2J+3, I+J+5) = …
                        … = A' (2I+J-1, 3I+1)
    ENDDOALL
ENDDOALL
/*Area3: region which must be executed serially */
DO I = l1, u1
  DO J = max(2I-8, l2), min(2I-6, u2 )
    IF ((2I-J-8≠0 & 2I-J-6≠0)) & (( ( J<24-2I)) & (J>(3+2I)/4)) then
    A(2J+3, I+J+5) = …
                    … = A(2I+J-1,3I+1)
  ENDDO
ENDDO
/*Area2: region which can be executed in parallel */
DOALL I = l1, u1
  DOALL J = max(l2, 2I-8), u2
    IF((J < 2I-6) | ( J < (3+2I) / 4))
    A(2J+3, I+J+5) = …
                    … = A(2I + J - 1, 3I + 1)
  ENDDOALL
ENDDOALL
```

(b) Transformed loop after the OTRP mechanism.

Fig. 4. Partitioning a loop of Example 2.

Array A elements in the first loop may be used before they are defined due to the presence of an anti-dependence. We rename array A by copying array A into array A′ to remove anti-dependences based on Lemma 7. Hence, the loop can be executed in parallel. In $Area_3$, there will be both flow dependence head and tail sets. The loop in this Area should be executed in serial. Because all the flow dependence tails have been executed in $Area_2$, which is a flow dependence unique head set, the loop can be executed in parallel according Theorem 2. To summarize, we give a formal algorithm for the OTRP mechanism in Fig. 5 and show how our proposed OTRP region partitioning technique and nested loop transformation algorithm are applied to the original loop to obtain a transformed loop. On the other hand, the OTRP mechanism can also be applied to an n-nested loop when there exists a k-dimensional array, where $n < k$. If $n < k$, then all the information needed by the OTRP mechanism can be obtained since the number of variables is less than the number of diophantine Eqs. However, if $n > k$, then OTRP can be applied only when there are fewer than k variables in the diophantine Eqs.

Algorithm *Optimized Three Region Partitioning*
(Transformation of a two-level nested loop with non-uniform dependences)
Input: a two-level nested loop with non-uniform dependences.
(1) Partitioning
- Step 1: Find the linear Diophantine Eq. in the two-level nested loop with non- uniform depen dences according to the formal expressions in Eqs. (1) and (2).
- Step 2: Solve the linear Diophantine Eq. and use the inequalities of the nested loop before conducting transformation to find the dependence convex hull DCH_1 and DCH_2 according to the formal expressions of Eqs. (3) and (4), respectively.
- Step 3: Find the dependence vector sets of DCH_1 and DCH_2 according to the formal expressions of Eqs. (5) and (6), respectively
- Step 4: Partition the iteration space into three areas according to Theorems 1, 2, and 3 with DCH_1, DCH_2, $d_i(x, y) = 0$ and $d_j(x, y) = 0$.

(2) Loop Transformation
- Step 1: Perform copy-renaming if there exists anti-dependence in the area.
- Step 2: Extract inequalities according to the edge of DCH_1 and DCH_2 and the line of $d_i(x, y) = 0$ and $d_j(x, y) = 0$.
- Step 3: Find the absolute maximum and minimum for each loop index.
- Step 4: Transform the indices.
- Step 5: Calculate the nested loop bounds.
- Step 6: Replace the arrays with renamed one if copy-renaming is applied.

Fig. 5. Algorithm of the OTRP mechanism.

4. COMPARISON WITH PREVIOUS WORK

The OTRP scheme is similar to the ITRP and the three region partitioning methods in the sense that the iteration space is divided into two parallel regions and one serial

region. The size of the parallel region in the OTRP technique is not less than that in the three region partitioning or the ITRP mechanism. To show the differences among OTRP, ITRP, and unique set oriented partitioning, we add unique set oriented partitioning as a reference tool. Both the OTRP and ITRP mechanisms divide the iteration space into three regions, and the execution order for the partitions is always the same($Area_1 \rightarrow Area_3 \rightarrow Area_2$). However, the serial region of the ITRP scheme is much larger than that of the OTRP scheme. We will compare the advantages and disadvantages of the four mechanisms based on five cases of the unique set oriented partitioning.

Case 1: There is only one kind of dependence, and DCH1 does not overlap with DCH2(Fig. 6).

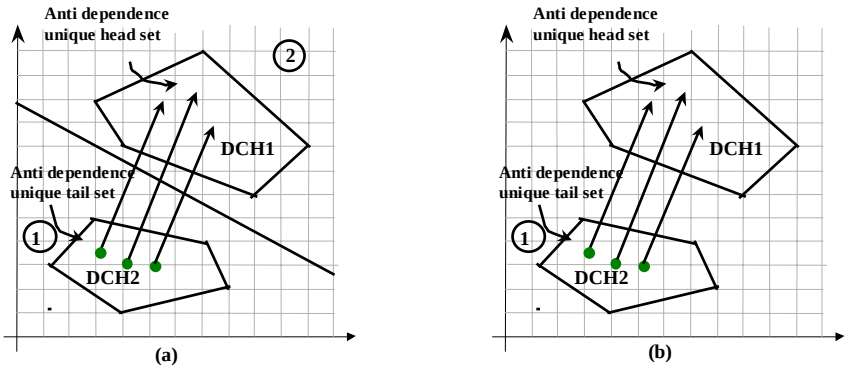


Fig. 6. In the case with one kind of dependence, DCH1 does not overlap with DCH2: (a) Two regions divided by unique set oriented partitioning. (b) One region divided by ITRP or OTRP.

In unique set oriented partitioning, any line drawn between DCH1 and DCH2 divides the iteration space into two areas. As shown in Fig. 6(a), the iterations within each area can be executed in parallel. However, the execution sequence for $Area_1$ and $Area_2$ is $Area_1 \rightarrow Area_2$. As shown in Fig. 6(b), because the copy-renaming technique is used in both the ITRP and OTRP schemes, only one area is partitioned, and only it can be executed in parallel. Thus, the parallelism degree for both the OTRP and ITRP mechanisms is twice that for the unique set oriented partitioning technique in this case.

Case 2: There is only one kind of dependence, and DCH1 overlaps with DCH2.

As illustrated in Fig. 7(a), DCH1 and DCH2 overlap, producing three distinct areas denoted by $Area_1$, $Area_2$ and $Area_3$, respectively. In unique set oriented partitioning, $Area_3$ contains both tail and heads of dependences, so iterations in this area must be executed in serial. $Area_1$ and $Area_2$ can be executed in parallel, and the execution order is $Area_1 \rightarrow Area_3 \rightarrow Area_2$. By means of copy-renaming, the total iteration space can be executed in parallel in both ITRP and OTRP schemes as shown in Fig. 7(b). Thus, the degree of parallelism in both the OTRP and ITRP schemes is much higher than that in the unique set oriented partitioning technique.

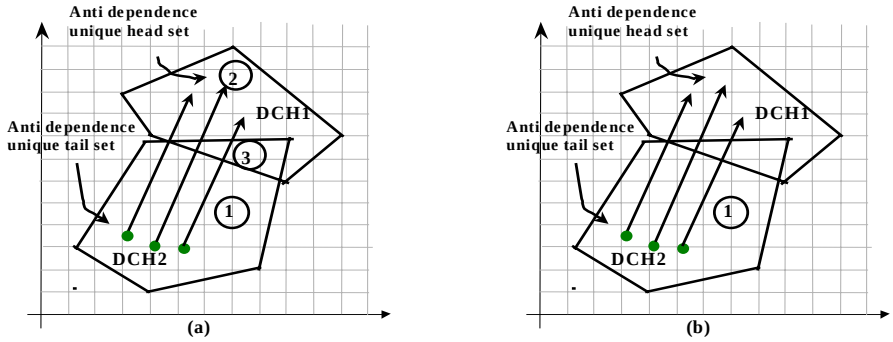


Fig. 7. In the case with one kind of dependence, DCH1 overlaps with DCH2: (a) Three regions divided by unique set oriented partitioning. (b) Only one region divided by ITRP or OTRP.

Case 3: There are two kinds of dependences and DCH1 does not overlap with DCH2.

As shown in Fig. 8(a), the iteration space is divided into four areas by unique set oriented partitioning. Because the unique head sets must be executed after the unique tail sets are executed, one probable order is $Area_1 \rightarrow Area_2 \rightarrow Area_3 \rightarrow Area_4$. The iterations in these four areas partitioned by this scheme can be executed in parallel. By implementing copy-renaming, the iteration space can be partitioned into two areas and executed in parallel in both ITRP and OTRP schemes as shown in Fig. 8(b). Also, as the flow dependence head sets have to be executed after the flow dependence tail sets are executed, the execution order has to be $Area_1 \rightarrow Area_2$. Thus, the degree of parallelism in both the ITRP and OTRP mechanisms is twice that in unique set oriented partitioning in this case.

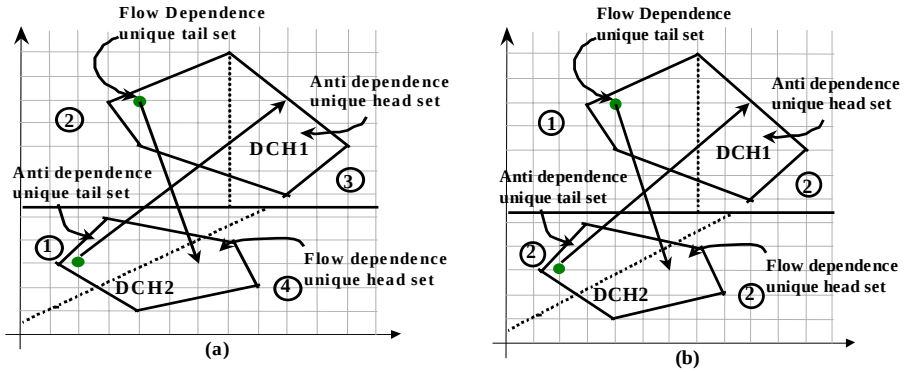


Fig. 8. In the case with two kinds of dependences, DCH1 does not overlap with DCH2: (a) Four regions divided by unique set oriented partitioning. (b) Two regions divided by ITRP or OTRP.

Case 4: There are two kinds of dependences, DCH1 overlaps with DCH2, and there is at least one isolated unique set.

As shown in Fig. 9(a), the iteration space is divided into six areas by unique set oriented partitioning. Because the unique head sets must be executed after the unique tail sets are executed, one probable order is $Area_1 \rightarrow Area_2 \rightarrow Area_5 \rightarrow Area_4 \rightarrow Area_6 \rightarrow Area_3$. The iterations in areas 1, 2, 3 and 4 partitioned by this scheme can be executed in parallel, and the iterations in areas 5 and 6 contain both tail and heads, so they must be executed in a serial manner. By implementing copy-renaming, the iteration space can be partitioned into three areas in the ITRP and OTRP schemes as shown in Figs. 9(b) and (c), respectively.

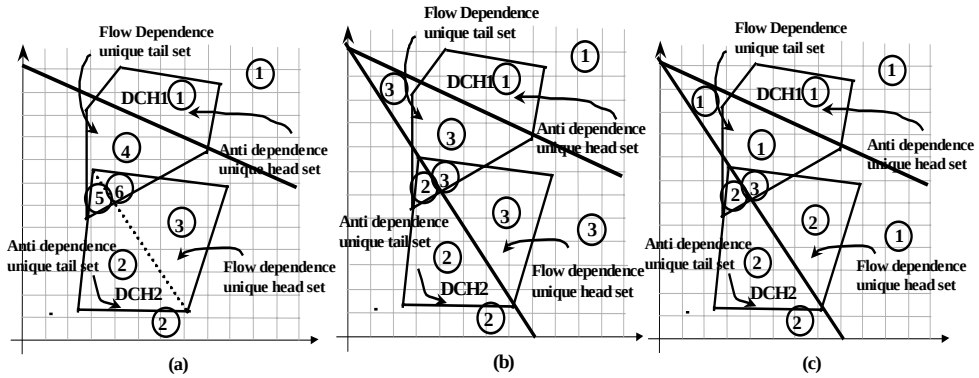


Fig. 9. With two kinds of dependences, DCH1 overlaps with DCH2, and there is at least one isolated unique set. (a) Six regions divided by unique set oriented partitioning. (b) Three regions divided by ITRP. (c) Three regions divided by OTRP.

Further more, as the flow dependence head sets must be executed after flow dependence tail sets are executed, the execution order must be $Area_2 \rightarrow Area_3 \rightarrow Area_1$. In Fig. 9(a), the iterations in $Area_2$ and $Area_1$ contain only unique sets, so they can be executed in parallel. However, the iterations in $Area_3$ contain both tail and heads; they have to be executed in a serial manner as shown in Fig. 9(b) and Fig. 9(c). Also, as $Area_3$ must be executed in serial in both the ITRP and OTRP mechanisms, the parallelism has to be bounded by the number of iterations in $Area_3$. From Figs. 9(b) and (c), we can conclude that the OTRP scheme is more effective than the ITRP mechanism due to the smaller size of the serial region $Area_3$.

Case 5: There are two kinds of dependences, and all the unique sets overlay each other. As shown in Fig. 10(a), the iteration space is divided into four areas by unique set oriented partitioning. As the unique head sets must be executed after the unique tail sets are executed, one probable order is $Area_1 \rightarrow Area_3 \rightarrow Area_4 \rightarrow Area_2$. The iterations in areas 1 and 2 partitioned by this scheme can be executed in parallel. Since the iterations in areas 3 and 4 contain both tail and heads, they have to be executed in a serial manner. By implementing copy-renaming, the iteration space can be partitioned into three areas in the ITRP and OTRP schemes as shown in Figs. 10(b) and (c), respectively.

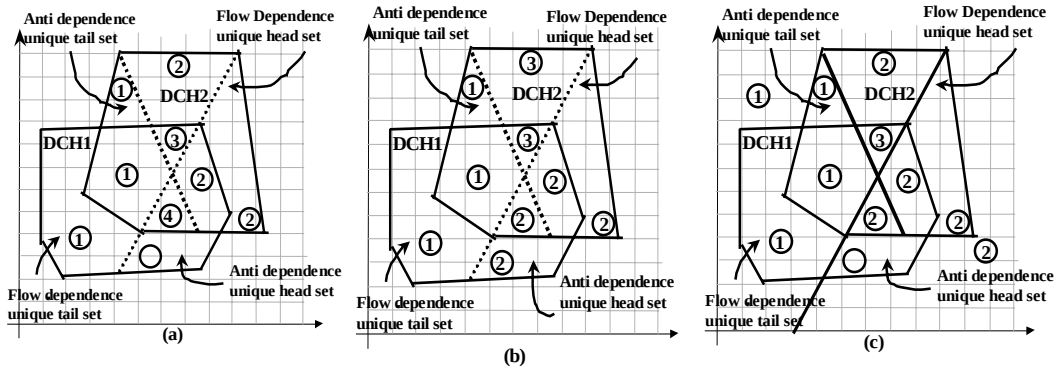


Fig. 10. There are two kinds of dependences, and all unique sets overlay each other. (a) Four regions divided by unique set oriented partitioning. (b) Three regions divided by ITRP. (c) Three regions divided by OTRP.

Because the flow dependence head sets must be executed after flow dependence tail sets are executed, the execution order has to be $Area_1 \rightarrow Area_3 \rightarrow Area_2$. In Fig. 10(a), the iterations in $Area_2$ and $Area_1$ contain only unique sets, so they will be executed in parallel. However, the iterations in $Area_3$ contain both tail and head; they must be executed serially as shown in Figs. 10(b) and (c). Because the $Area_3$ must be executed serially in both the ITRP and OTRP mechanisms, the degree of parallelism of the whole iteration space will be bounded by the number of iterations in $Area_3$. Based on Figs. 10(b) and (c), we can conclude that the OTRP scheme is more effective than the ITRP mechanism because the size of $Area_3$ in the ITRP is much larger than that in OTRP scheme.

In all the cases of unique set partitioning, OTRP extracts more parallelism than either unique set oriented partitioning or ITRP do. We will next present preliminary performance evaluations to demonstrate the effectiveness of our OTRP scheme in the next section.

5. PRELIMINARY EXPERIMENTAL RESULTS

The OTRP method is similar to ITRP and the three region partitioning in the sense that the iteration space is divided into two parallel regions and one serial region. The size of the parallel region in OTRP is not less than that in ITRP or in the three region partitioning method mentioned above.

In unique set oriented partitioning, it is a complicated task to divide the iteration space into up to eight partitions, and it is also difficult to determine the best execution order for the partitions. On the other hand, in the ITRP and OTRP mechanisms, it is simple to divide the iteration space into three regions, and the execution order for the partitions is always the same ($Area_1 \rightarrow Area_3 \rightarrow Area_2$). In some cases (case 4 and case 5), ITRP has a much larger serial region than does the OTRP scheme.

Theoretical speedup based on performance analysis can be represented as follows.

If an unlimited number of processors is assumed, then each parallel partition, executed in one time step. Hence, the total execution time will be equal to the number of parallel partitions, N_p, plus the number of sequential iterations, N_s, plus some synchronization, scheduling and copy-renaming overheads. Generally speaking, the speedup can be represented by the ratio of the total sequential execution time to the execution time on a parallel computer system as follows:

$$\text{Speedup} = (N_I * N_J) / (N_p + N_s + \zeta),$$

where N_I and N_J are the size of loops I and J, respectively. ζ is the combined overhead of synchronization, scheduling and copy-renaming. If we ignore the synchronization, scheduling and copy-renaming overheads, the speedup will be expressed as

$$\text{Speedup} = (N_I * N_J) / (N_p + N_s).$$

Fig. 11(a) shows one probable partitioning case of unique set oriented partitioning. $Area_2$ contains only flow dependence heads, and $Area_1$ contains flow dependence tails and anti-dependence tails as well as heads. By means of minimum dependence distance tiling, $Area_1$ can be tiled with width = 1 and height = 4. Thus, 24 tiles are obtained. The speedup for this method is (10*11)/(24 + 1) = 4.4 as shown in Fig. 11(b). On the other hand, if ITRP or OTRP is applied to this example, then the maximum speedup will be achieved, which is

$$\text{Speedup} = (N_I * N_J) / (N_p + N_s) = (10*11)/(2 + 0) = 55.$$

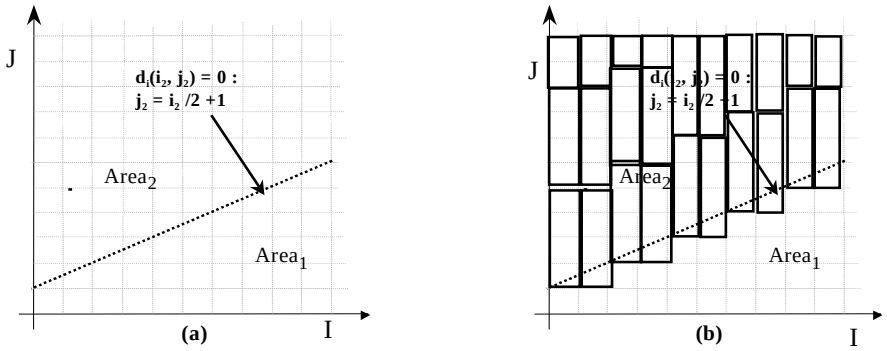


Fig. 11. An example of (a) Unique set oriented partitioning with one parallel area ($Area_1$) and one serial area ($Area_2$). (b) The serial area is partitioned by the minimum dependence distance tiling scheme.

Let us consider the loop shown in Example 2. Three region partitioning can divide the iteration space into one parallel region, $Area_1$, and one serial region, $Area_3$, as shown in Fig. 12. The speedup can be computed as (10*10) / (1 + 60) = 1.6. If dependence uniformization is applied to $Area_3$, only a limited amount of speedup is obtained. Hence, if the serial region is large, the performance of the loop will be significantly de-

graded. Unique set oriented partitioning will also divide the iteration space into two parts according to the line $j_2 = 4i_2 - 10$. $Area_1$ will be on the left side of the line, and $Area_2$ will be the right side of the line. The execution order will be $Area_1 \rightarrow Area_2$, and $Area_2$ should be tiled by means of minimum dependence distance tiling. $Area_2$ will be tiled into 34 tiles with width = 1 and height = 2. Thus the speedup achieved by this method will be (10*10) / (1 + 34) = 2.9.

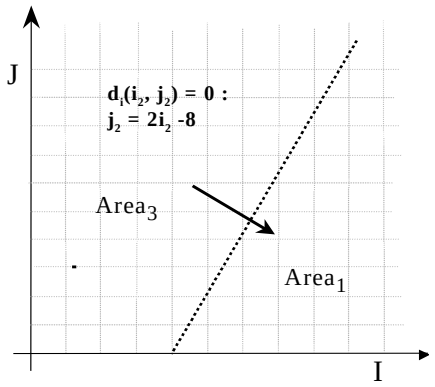


Fig. 12. Examples of region partitioning.

ITRP divides the iteration space into two parallel regions and one parallelizable region as shown in Fig. 13(b). The execution order will be $Area_1 \rightarrow Area_3 \rightarrow Area_2$. Without using the variable size partitioning technique of ITRP, $Area_3$ can be partitioned into 10 partitions. Hence, the speedup that con be obtained by this method is (10*10) / (1 + 10 + 1) = 8.3. The OTRP mechanism divides the iteration space into two parallel regions and one parallelizable region as shown in Fig. 13(c). The execution order is $Area_1 \rightarrow Area_3 \rightarrow Area_2$. Without using the variable size partitioning technique of ITRP, $Area_3$ will be partitioned into 8 partitions by means of the minimum dependence distance tiling technique. Hence, the speedup for the OTRP method will be (10*10) / (1 + 8 + 1) = 10.

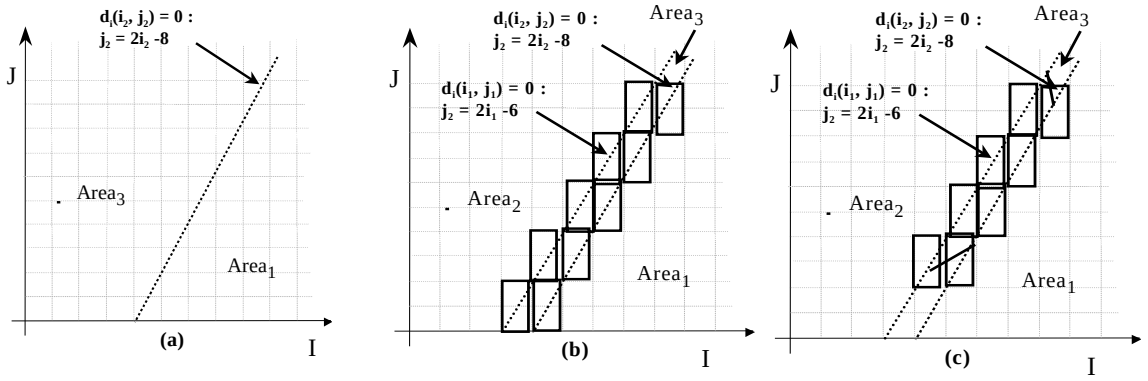
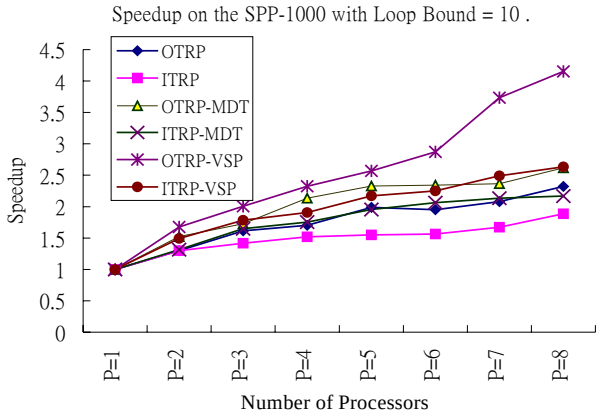


Fig. 13. Divided by the (a) three region partitioning scheme, (b) ITRP scheme, (c) OTRP scheme.

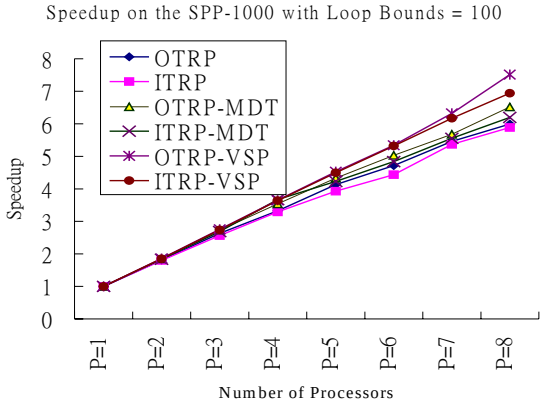
If the variable size partitioning technique of ITRP is employed, $Area_3$ can be partitioned into 4 partitions. Hence, the speedup that can be achieved by ITRP is (10*10) / (1 + 4 + 1) = 16.7. On the other hand, if the variable size partition in the OTRP mechanism is used, $Area_3$ can be partitioned into 2 partitions. Hence, the speedup that can be achieved with this method is (10*10) / (1 + 2 + 1) = 25. It is clear that OTRP exploits more parallelism than does either the three-region partitioning or unique set oriented partitioning method. In addition, the OTRP scheme extracts much more parallelism than the ITRP scheme and other mechanisms do. In order to see whether our method can be implemented and whether our method can find more parallelism than other existing popular methods can, we implemented our method in a parallel compilation environment and ran it on a real machine to evaluate its performance.

We have constructed the OTRP mechanism in the SUIF [13] parallel compilation environment, run experiments on an IBM SPP1000 with 8 processors and used the EXPA system to analyze the performance. EXPA is a tool developed by SPP to accurately measure and graphically display the tasking performance for a job run on an arbitrarily loaded SPP system. We used the User-directed tasking directives #pragma _CNX to construct our fully parallelizable and critical section in the iteration space. We used Example 2 to evaluate different mechanisms. Fig. 14 shows the speedup achieved by our technique, OTRP and ITRP combined with the MDT (Minimum Dependence distance Tiling) and VSP (Variable Size Partitioning) techniques. Our technique delivered better performance when the loop bounds of this example were set to 10 and 100 as shown in Figs. 14(a) and (b). As the number of processor increased, the performance of OTRP combined with the VSP mechanism became even better than that of the others. However, the speedup degree of different mechanisms was nearly linear as shown in Fig. 14(c) due to the massive parallelism degree and the limited number of processors in the IBM SPP-1000 when the loop bound was set to 1000.

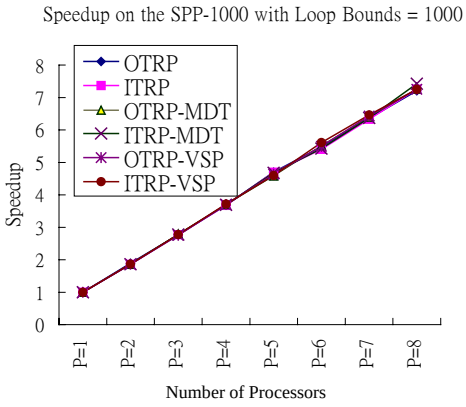


(a) Speedup on the SPP-1000 with Loop Bound = 10

Fig. 14. (a) Evaluation of the program in Example 2 on a real machine.



(b) Speedup on the SPP-1000 with Loop Bound = 100

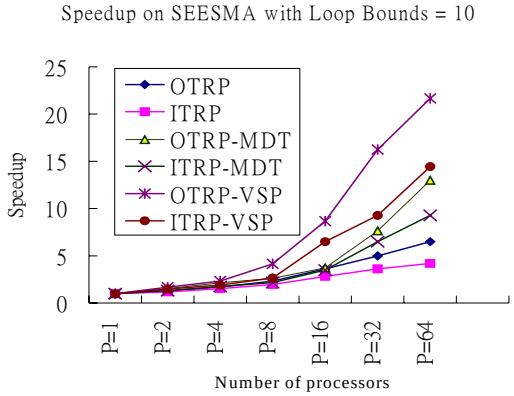


(c) Speedup on the SPP-1000 with Loop Bound = 1000

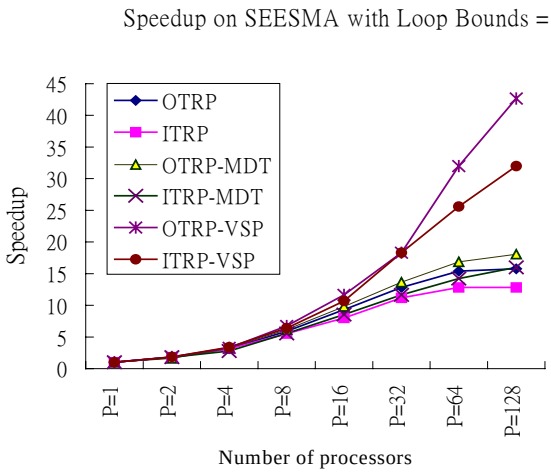
Fig. 14. (Cont'd) Evaluation of the program in Example 2 on a real machine.

In order to see whether our OTRP technique could perform better than the existing region partitioning mechanisms on a system with a large number of processors, we construct a multiprocessor evaluation environment in order to measure their performance. Different mechanisms were implemented, and the object code was evaluated on a simulator called SEESMA (A Simulation and Evaluation Environment for Shared-Memory Multiprocessor Architecture) [17], which was enhanced from MINT [18]. This system is a highly parallel shared memory multiprocessor system environment. It is a CC-NUMA architecture consisting of many clustering nodes interconnected by the K-ary, n-Cube network. Each node contains a local shared-memory area, a cache, a processor environment, and a local bus. Each processor environment includes a two-level cache hierarchy associated with write buffers. The cache hierarchy interfaces with the local portion of the shared memory.

Fig. 15 shows the speedup obtained by our technique, OTRP and ITRP combined with the MDT (Minimum Dependence distance Tiling) and VSP (Variable Size Partitioning) techniques in the SEESMA environment. Our technique delivered better performance when the loop bounds of this example were set to 10, 100 and 1000 as shown in Figs. 15(a), (b) and (c). As the number of processor increased to 128, the performance of OTRP combined with the VSP mechanism became even better than that of the others. We also found that the speedup achieved when the loop bound was set to 10 by ITRP-VSP and OTRP-VSP was 14.44 and 21.67, respectively, as shown in Fig. 15(a). These values are closer to the ideal speedup values of 16.7 and 25, respectively, than were the speedup values achieved on the IBM SPP-1000, which were 2.5 and 4.2 as shown in Fig. 14(a). These results could be obtained because more parallelism can be achieved with more processors.

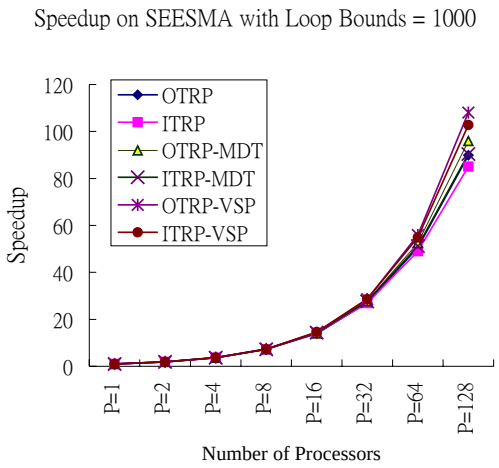


(a) Speedup on SEESMA with Loop Bounds = 10



(b) Speedup on SEESMA with Loop Bounds = 100

Fig. 15. Evaluation on the SEESMA environment for the program of Example 2.



(c) Speedup on SEESMA with Loop Bounds = 1000

Fig. 15. (Cont'd) Evaluation on the SEESMA environment for the program of Example 2.

6. CONCLUSIONS AND FUTURE WORK

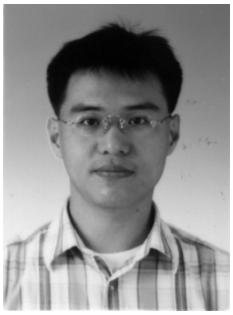
In this paper, we have studied the problem of transforming nested loops with non-uniform dependences in order to maximize parallelism by means of a new scheme, called OTRP that is, based on convex hull theory. OTRP can easily divide the iteration space into two parallel regions and one parallelizable region. In addition, the small size of the serial region in OTRP produces more parallelism than do the currently available schemes. On the other hand, the OTRP scheme can partition the areas easily when the DCH bounds or dependence vector's lines are known in advance. In comparison with the other partitioning methods that are based on convex hull theory, OTRP has several advantages, such as simple partitioning and high speedup. We implemented our scheme in the SUIF parallel compilation environment and evaluated it both on a real machine and in a simulation environment. We also ran it with large loop bounds to simulate the case of large scientific benchmarks. Under all the above conditions, our scheme performed better than the other existing mechanisms.

In the future, we will improve the OTRP scheme by means of higher dimensional nested loops and loops with multiple dependences. We also think that scheduling and allocation are needed for massive parallel iterations derived using the OTRP mechanism when the system overhead becomes large. On the other hand, the parallelizable part divided by the OTRP scheme can also be enhanced by using more aggressive methods.

REFERENCES

1. Z. Shen, , Z. Li, and P. C. Yew, "An empirical study on array subscripts and data dependencies," in *Proceedings of the 1989 ICPP IEEE*, 1989, pp. 145-152.

2. T. H. Tzen and L. M. Ni, "Dependence uniformization: A loop parallelization Technique," *IEEE Transactions on Parallel and Distributed System*, Vol. 4, 1993, pp. 547-558.
3. H. Zima, *Supercompilers for Parallel and Vector Computers*, Addison Wesley, 1990.
4. S. Punyamurtula and V. Chaudhary, "Minimum dependence distance tiling of nested loops with non-uniform dependences," in *Proceedings of 6th IEEE Symposium on Parallel and Distributed Processing*, 1994, pp. 74-81.
5. A. Zaafrani and M. Ito, "Parallel region execution of loops with irregular dependences," in *Proceedings of the International Conference on Parallel Processing*, 1994, pp. 11-19.
6. J. Ju and V. Chaudhary, "Unique sets oriented partitioning of nested loops with non-uniform dependences," *International Conference on Parallel Processing*, 1996, pp. 45-52.
7. C. K. Cho and M. H. Lee, "A loop parallelization method for nested loops with non-uniform dependences," *International Conference on Parallel Processing*, 1997, pp. 314-321.
8. U. Banerjee, *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers, 1988.
9. M. J. Berger, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Transactions on Computers*, Vol. C-36, No. 5, 1987, pp. 570-580.
10. S. Y. Tseng, C. T. King, and C. Y. Tang, "Minimum dependence vector set: a new compiler technique for enhancing loop parallelism," in *Proceedings of 1992 International Conference on Parallel and Distributed Systems*, 1992, pp. 340-346.
11. W. Shang, E. Hodzic, and Z. Chen, "On uniformization of affine dependence algorithms," *IEEE Transactions on Computers*, Vol. 7, 1996, pp. 827-840.
12. D. K. Chen and P. C. Yew, "On effective execution of nonuniform DOACROSS loops," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, 1996, pp. 463-476.
13. R. Wilson, M. S. Lam, and J. Hennessy, *An Overview of the SUIF Compiler System*, Computer Systems Lab, Stanford University, 1996.
14. S. Y. Tseng, C. T. King, and C. Y. Tang, "Profiling dependence vectors for loop parallelization," in *Proceedings of 1996 IPPS*, pp. 23-27.
15. S. Punyanurtula, V. Chaudhary, J. Ju, and S. Roy, "Compile time partitioning of nested loop iteration spaces with non-uniform dependences," *Parallel Algorithms and Applications*, Vol. 12, No. 1-3, 1997, pp. 113-141.
16. M. J. Hwu and D. J. Buehrer, "An improved scheme for effective execution of nested loops with irregular dependence constraints," *Journal of the Chinese Institute of Electrical Engineering*, Vol. 2, No. 2, 1995, pp. 107-118.
17. J. P. Su, C. C. Wu, and C. Chen, "Reducing the overhead of migratory-shared access for the linked-based directory coherent protocols in shared memory multiprocessor systems," in *Proceedings of ICS'96 on Computer Architecture*, 1996, pp. 160-167.
18. J. E. Veenstra and R. J. Fowler, "MINT tutorial and user manual," Technical Report, 452, The University of Rochester, New York, 1994.



Der-Lin Pean (平德林) is a Ph.D. candidate in Computer Science and Information Engineering at National Chiao Tung University, Taiwan R.O.C. He received his B.S. degree in Information and Computer Engineering from Chung Yuan Christian University, Taiwan, R.O.C. He has served as a lecturer in the Department of Computer and Information Engineering as well as in Employment and Vocational Training Administration, the Council of Labor Ministry, Taiwan R.O.C. His current research interests include computer architecture, personal computer system architecture design, parallel processing system design, parallelizing compiler techniques, and microprocessor system design.

Cheng Chen (陳正) is a professor in the Department of Computer Science and Information Engineering at National Chiao Tung University, Taiwan, R.O.C. He received his B.S. degree from the Tatung Institute of Technology, Taiwan, R.O.C. in 1969 and his M.S. degree from the National Chiao Tung University, Taiwan, R.O.C. in 1971, both in electrical engineering. Since 1972, he has been on the faculty of National Chiao Tung University, Taiwan, R.O.C. From 1980 to 1987, he was a visiting scholar at the University of Illinois at Champaign–Urbana. During 1987 and 1988, he served as the chairman of the Department of Computer Science and Information Engineering at National Chiao Tung University. From 1988 to 1989, he was a visiting scholar at Carnegie Mellon University (CMU). Between 1990 and 1994, he served as the deputy director of the Microelectronics and Information Systems Research Center (MIRC) at National Chiao Tung University. His current research interests include computer architecture, parallel processing system design, parallelizing compiler techniques, and high performance video server design.