Indian Journal of Science and Technology, Vol 8(13), DOI: 10.17485/ijst/2015/v8i13/72684, July 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Loop Splitting Method of Irregular and Flow Dependence Loops

Sam-Jin Jeong*

Division of Information and Communication Engineering, Baekseok University, Korea; sjjeong@cheonan.ac.kr

Abstract

A lot of work has been done in parallelizing nested loops with uniform dependences, from dependence analysis to loop transformation. Loops with non-uniform dependences are not so uncommon in the real world. This paper proposes an efficient method of splitting and transforming nested loops with irregular and flow dependences for maximizing parallelism. Our approach is based on the Convex Hull theory that has adequate information to handle irregular dependences, and also based on minimum dependence distance tiling methods. We will first show how to find the incrementing minimum dependence distance. Next, we will propose how to tile the iteration space efficiently according to the incrementing minimum dependence distance. Finally, we will show how to achieve more parallelism by loop interchanging and how to transform it into parallel loops. Comparison with some other methods shows more parallelism than other existing methods.

Keywords: Flow Dependence, Irregular Dependence, Loop Transformation, Parallelizing Compiler

1. Introduction

Automatic transformation of a sequential program into a parallel form is a subject that presents a great intellectual challenge, and at the same time promises a large practical reward. On one hand, there is a tremendous investment in existing sequential programs that we would like to run with ever increasing speed. On the other hand, scientist and engineers continue to write their application programs in sequential languages, and keep demanding higher and higher speedups. Over a period of several decades, much research has been done on using restructuring compilers to enable the parallel execution of sequential programs[1].

Example l.

```
do i = 1, 10
   do j = 1, 10
      A(i+1, j) = . . .
      . . . = A(i, j-1)
   enddo
enddo
```

Example 2.

```
do i = 1, 10
   do j = 1, 10
      A(2*i+3, j+1) = . . .
      . . . = A(2*j+i+1, i+j+3)
         enddo
enddo
```

Example 3.

```
do i = 1, 10
   do j = 1, 10
      A(2*i+ j+1, i+j+3) = . . .
         . . .= A(i+4, 2*j+3)
          enddo
enddo
```

According to an empirical study[2], nearly 66% of the array references have linear or partially linear subscript expressions and 45% of two dimensional array references are coupled and most of these lead to irregular dependences.

Example 1 is loop with uniform dependences. The dependence vector of Example 1 is (1, 1) as shown in Figure 1(a). In the same fashion, we call some dependences non-uniform when dependence vectors are in irregular and complex patterns that cannot be expressed by distance vector. Figure 1(b) shows the dependence patterns of Example 2 in the iteration space. An example given in Example 3 illustrates the case that there is irregular and flow dependence. Figure 1(c) shows the dependence patterns of Example 3.

*Author for correspondence

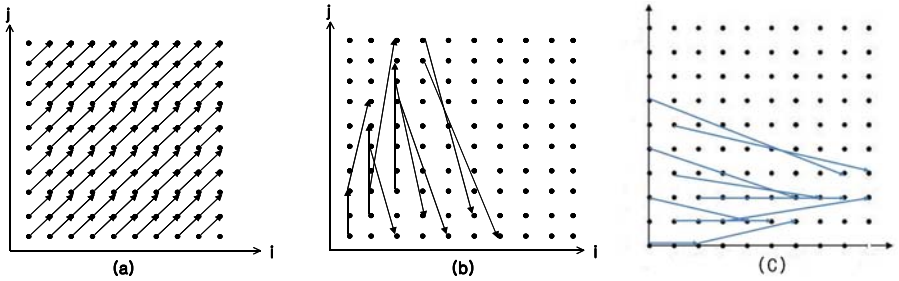

Figure 1. Iteration spaces of (a) Example 1, (b) Example 2, (c) Example 3.

This paper will focus on parallelizing perfectly nested loops with irregular and flow dependences.

2. Data Dependence Analysis in Irregular and Flow Dependence Loop

The loop model considered in this paper is doubly nested loops in Figure 2, where $f_1(I, J)$, $f_2(I, J)$, $f_3(I, J)$, and $f_4(I, J)$ are linear functions of loop variables. Both lower and upper bounds for loop variables should be known at compile time.

The loop carries cross iteration dependences if and only if there exist four integers (i_1, j_1, i_2, j_2) satisfying the system of linear diophantine equations given by (1) and the system of inequalities given by (2). The general solution to these equations can be computed by the extended GCD[8] and forms a **DCH**.

The loop model considered in this paper is doubly nested loops with linearly coupled subscripts and both lower and upper bounds for loop variables should be known at compile time. The loop model has the form in Figure 2.

The dependence distance function $d(i_1, j_1)$ in flow dependence loops gives the dependence distances $d_i(i_1, j_1)$ and $d_j(i_1, j_1)$ in dimensions i and j, respectively. For uniform dependence vector sets these distances are constant. But, for the irregular dependence sets these distances

```
do i = l1, u1
    do j = l2, u2
      A(a11 i + b11 j + c11, a12 i + b12 j + c12) = ...
            ... = A(a21 i + b21 j + c21, a22 i + b22 j + c22)
    enddo
enddo
```

Figure 2. A doubly nested loop model.

are linear functions of the loop indices. We can write these dependence distance functions in a general form as

$$d(i_1, j_1) = (d_i(i_1, j_1), d_j(i_1, j_1)) \quad (1)$$

$$d_i(i_1, j_1) = p_1^* i_1 + q_1^* j_1 + r_1$$

$$d_j(i_1, j_1) = p_2^* i_1 + q_2^* j_1 + r_2$$

where p_i, q_i, and r_i are real values and i_1 and j_1 are integer variables of the iteration space.

The dependence distance function $d(i_1, j_1)$ can be represented by direction of the dependence. Figure 3 shows all possible flow dependence directions in doubly nested loops.

The dependence direction is written as $d = (i, j)$, where i is the value of $d_i(i_1, j_1)$ and j is the value of $d_j(i_1, j_1)$. From Figure 3, we know that there are possible flow dependence directions as follows.

ⓐ $\boldsymbol{i} = 0$ and $\boldsymbol{j} > 0$,

ⓑ $0 < \boldsymbol{i} < \boldsymbol{j}$,

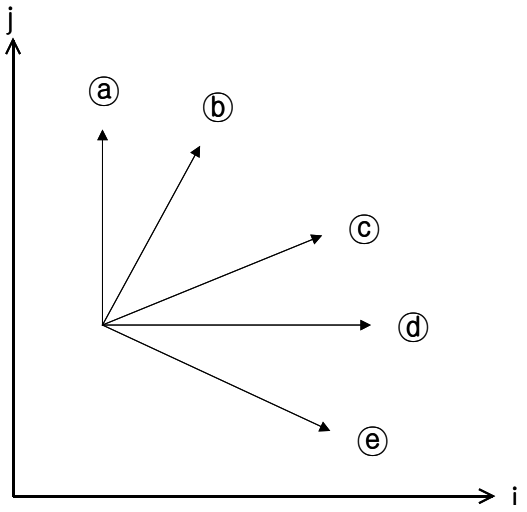

Figure 3. Possible flow dependence directions.

ⓒ $i > j > 0$,
ⓓ $i > 0$ and $j = 0$,
ⓔ $i > 0$ and $j < 0$

Cases ⓐ ~ ⓓ are flow dependence directions that show loop interchanging is possible. In case ⓔ, loop interchanging is possible for nested loops with uniform dependences, but is impossible for nested loops with irregular dependences. In case i < j, such as in cases ⓐ and ⓑ, loop parallelization can be improved by loop interchanging.

The properties and theorems for splitting of nested loops with flow dependence can be described as follows.

Theorem 1: *If there is only flow dependence in the loop, DCH1 contains flow dependence tails and DCH2 contains flow dependence heads.*

Theorem 2: *If there is only flow dependence in the loop, then $d_i(x, y) = 0$ or $d_j(x, y) = 0$ does not pass through any DCH.*

If there exists only flow dependence in the loop, then $d_i(x_1, y_1) = 0$ or $d_j(x_1, y_1) = 0$ does not pass through any IDCH (Integer Dependence Convex Hull) because the IDCH is a subspace of DCH (Dependence Convex Hull)[5].

Theorem 3: *If there is only flow dependence in the loop, the minimum and maximum values of the dependence distance function $d(x_1, y_1)$ appear on the extreme points.*

Theorem 4: *If there is only flow dependence in the loop, the minimum dependence distance value d_{imin} is equal or greater than zero.*

From theorem 4, we know that when there is only flow dependence in the loop and d_{imin} is zero, d_{jmin} is greater than zero. In this case, since $d_j(x_1, y_1) = 0$ does not pass through the IDCH, the minimum value of $d_j(x_1, y_1)$, d_{jmin}, occurs at one of the extreme points.

Theorem 5: *If there is only flow dependence in the loop, the difference between the distance of a dependence and that of the next dependence, d_{inc}, is equal to or greater than zero.*

Thus, d_{inc} is equal to or greater than zero when there is only flow dependence in the loop.

3. Loop Spitting and Transformation for Flow Dependence

Cho and Lee[2] present a more general and powerful loop splitting method to enhance all parallelism on a single loop. The method uses more information from the loop such as increment factors, and the difference between the distance of dependence, and that of the next dependence.

The minimum dependence distance Tiling method[6] presents an algorithm to convert the extreme points with real coordinates to the extreme points with integer coordinates. The method obtains an IDCH from a DCH. It can compute d_{imin}, the minimum value of the dependence distance function $d_i(i_1, j_1)$ and d_{jmin}, the minimum value of the dependence distance function $d_j(i_1, j_1)$ from the extreme points of the IDCH. The first minimum dependence distances d_{imin} and d_{jmin} are used to determine the uniform tile size in the iteration space.

We will show cases that can tile the iteration space by real value p_i, q_i and r_i of equation (1) as follows.

Case 1: When $p_1 > 0$ and $q_1 \geq 0$

In this case, we know that the difference between the distance of dependence and that of the next dependence in loop with flow dependence, d_{inc}, is equal to or greater than zero.

For each i_1, d_{imin} is incremented as the value of i_1 is incremented. So, the second d_{imin} is equal to or greater than the first one, and the third one is greater than the second one, and so on.

The improved splitting method for doubly nested loops with irregular and flow dependence is described as Procedure Splitting_Method. The Procedure Splitting_Method shows the transformation of doubly nested loops satisfying the case that there is only flow dependence in the loop and $p_1 > 0$ and $q_1 \geq 0$. This algorithm computes the incrementing minimum dependence distance, tiles the iteration space efficiently according to the incrementing minimum dependence distance, and transforms it into parallel loops.

By the minimum dependence distance Tiling [Pun96], we can get the first source point, (i_1, j_1), which is one among the extreme points of the IDCH. From DCH1, we also get the dependence distance function $d_i(i_1, j_1)$. Given the first source point (i_1, j_1), the dependence distance function $d_i(i_1, j_1)$, and both lower and upper boundaries for loop variables, we can start Procedure Splitting_Method as follows.

In step 1 of the Procedure Splitting_Method, the first minimum dependence distance d_{imin} $(= p_1^* i_1 + q_1^* j_1 + r_1)$ is computed. Because d_{imin} is real value in doubly nested loop with irregular dependences, $d_{imin}(= Dist_1)$ is used as the minimum dependence distance. Next, St_2 and Tg_2,

i and j values for the target of the first dependence in the first tile, are computed. From loop in Figure 2, a general equation that computes the j_2 value for the target of the first iteration is as follows.

If (b_{21} == 0) then $j_2 = (a_{12}{}^*i_1 + b_{12}{}^*j_1 + c_{12} - a_{22}{}^*i_2 - c_{22}) / b_{22}$;
Else $j_2 = (a_{11}{}^*i_1 + b_{11}{}^*j_1 + c_{11} - a_{21}{}^*i_2 - c_{21}) / b_{21}$

If i or j value for the target of the first minimum dependence is equal to or greater than the upper bounds of the outer loop or the inner loop, respectively, then GOTO step 4. Otherwise, GOTO step 3.

In step 2, the minimum dependence distance in the nth tile (= $Dist_n$) is computed. The i value of the target for the first iteration in each tile, $Sr_n + Dist_n$, is selected as the first iteration in the next tile, St_{n+1}. And j value for the target of the first dependence in the nth tile, Tg_{n+1}, is computed by value b_{21}. If St_{n+1} or Tg_{n+1} is equal to or greater than the upper bounds of outer loop or inner loop, respectively, then GOTO step 4.

In step 3, i value, Sr_{n+1}, for the source of the first dependence in the next tile is obtained. The value q, difference between i value for the source of the first dependence and i value for the first iteration in the same tile, will be maximized parallelism from a loop.

In step 4, the original loop is transformed into n parallel tiles.

Case 2: When p1 > 0 and q1 < 0
Case 3: When p1 = 0 and q1 > 0
Case 4: When p1 = 0 and q1 = 0

In case 2 ~ 4, we can group the iterations among the dimension *i* into tiles of width of the first d_{imin}. Each tile is fully parallel, and tiles are executed in serial. In all cases, if $d_{jmin} > d_{imin}$, the outer loop *i* and the inner loop *j* can be interchanged for maximizing the size of the tile.

Figure 4(a) shows CDCH (Complete Dependence Convex Hull) of Example 3. As the example, we can obtain the following results using the improved splitting method proposed in this section.

From the algorithm to compute a two-dimensional IDCH in [5], we can obtain the extreme points such as (1, 1), (1, 10), and (5, 1) for DCH1, and the extreme points such as (1, 3), (10, 4), and (10, 2) for DCH2 as shown in Figure 4(a). The first minimum value of $d_i(i_1, j_1)$ occurs at one of the extreme points. The *i* value for the source of the first dependence in the second tile is 3. The *i* value in the third tile is 7. Then, we can divide the iteration space by three tiles as shown in Figure 4(b).

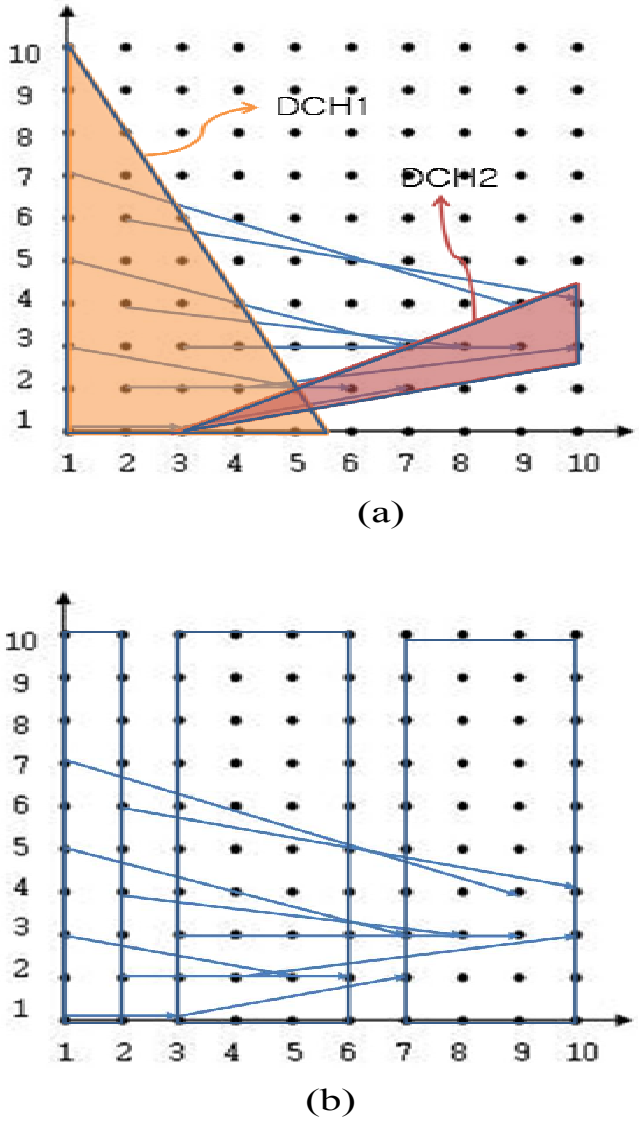

Figure 4. (a) CDCH, (b) The improved splitting method in Example 3.

4. Performance Analysis

This section discusses the performance analysis of our proposed methods through the comparisons with related works theoretically. Theoretical speedup for performance analysis can be computed as follows. Ignoring the synchronization, scheduling and variable renaming overheads, and assuming an unlimited number of processors, each partition can be executed in one time step. Hence, the total time of execution is equal to the number of parallel regions, *Np*, plus the number of sequential iterations, *Ns*. Generally, speedup is represented by the ratio of total sequential execution time to the execution time on

parallel computer system as follows:

$$Speedup = (N_i * N_j)/(N_p + N_s)$$

where N_i, N_j are the size of loop i, j, respectively

We will compare our proposed methods with the minimum dependence distance tiling method and the unique set oriented partitioning method as follows:

Let's consider the loop shown in Example 3. Figure 4(a) shows original partitioning of Example 3. This example is the case that there is only flow dependence and DCH1 overlaps DCH2. Applying the unique set oriented partitioning to this loop illustrates case 2 of [4]. This method can divide the iteration space into three regions: three parallel regions, AREA1 and AREA2, and one serial region, AREA3, as shown in Figure 5. The speedup for this method is (10*10)/(2+4) = 16.6.

Applying the minimum dependence distance tiling method to this loop illustrates case 1 of this technique[5], which is the case that line $d_i(i, j) = 0$ does not pass through the IDCH. The minimum value of $d_i(i, j)$, d_{imin}, occurs at the extreme point (1, 1) and $d_{imin} = 2$. The space can be tiled with width = 2, thus 5 tiles are obtained. The speedup for this method is (10*10)/5 =20.

Let's apply our proposed method - the improved splitting method as given in section 3. This loop is tiled by three areas as shown in Figure 4(b). The iterations within each area can be fully executed in parallel. So, the speedup for this method is (10*10)/3 = 33.3.

5. Conclusion

In this paper, we have studied the problem of transforming nested loops with irregular and flow dependences to maximize parallelism. Several methods are proposed in order to parallelize loops with non-uniform dependence. These techniques do a good job for some particular types of loops, but most such techniques perform some other types of loops poorly due to irregular and complex dependence constraints.

When there is only flow dependence in the loop, we propose the improved splitting method. The minimum dependence distance tiling method tiles the iteration space by the first minimum dependence distance uniformly. Our proposed method, however, tiles the iteration space by minimum dependence distance values that are incremented as the value of i_1 is incremented.

In comparison with some previous partitioning methods, the improved splitting method gives much better speedup than the minimum dependence distance tiling method and the unique set oriented partitioning method in the case that there is irregular and flow dependence. Our future research work is to develop a method for improving parallelization of higher dimensional nested loops.

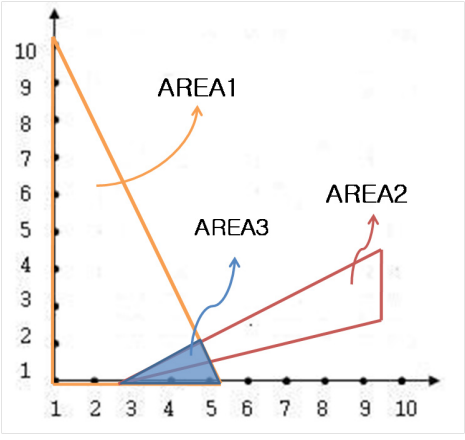

Figure 5. Regions of the loop partitioned by the unique sets oriented partitioning in Example 3.

6. Acknowledgement

This research is supported by 2015 Baekseok University Research fund.

7. References

1. Banerjee U. Loop Transformations for Restructuring Compilers: The Foundations. Norwell, Massachusetts: Kluwer Academic Publishers; 1993.
2. Cho CK, Shim JC, Lee MH. A loop transformation for maximizing parallelism from single loops with non-uniform dependences. Proceedings of High Performance Computing Asia '97; 1997. p. 696–9.
3. Cho CK, Lee MH. A loop parallization method for nested loops with non-uniform dependences. Proceedings of the International Conference on Parallel and Distributed Systems; 1997; p. 314–21.
4. Ju J, Chaudhary V. Unique sets oriented partitioning of nested loops with non-uniform dependences. Proceedings of International Conference on Parallel Processing; 1996. p. 45–52.
5. Punyamurtula S, Chaudhary V. Minimum dependence distance tiling of nested loops with non-uniform dependences. Proceedings of Symposium on Parallel and Distributed Processing; 1994. p. 74–81.
6. Punyamurtula S, Chaudhary V, Ju J, Roy S. Compile time partitioning of nested loop iteration spaces with

non-uniform dependences. Journal of Parallel Algorithms and Applications. 1996.
7. Tzen T, Ni L. Dependence uniformization: A loop parallelization technique. IEEE Transactions on Parallel and Distributed Systems. 1993; 4(5):547–58.
8. Zaafrani AA, Ito MR. Parallel region execution of loops with irregular dependences. Proceedings of the International Conference on Parallel Processing; 1994. p. 11–9.