http://dx.doi.org/10.15207/JKCS.2015.6.3.051

Extended Three Region Partitioning Method of Loops with Irregular Dependences

Sam-Jin Jeong[1*]
[1]Division of Information and Communication, Baekseok University

비규칙 종속성을 가진 루프의 확장된 세지역 분할 방법

정삼진[1*]
[1]백석대학교 정보통신학부

요 약 This paper proposes an efficient method such as Extended Three Region Partitioning Method for nested loops with irregular dependences for maximizing parallelism. Our approach is based on the Convex Hull theory, and also based on minimum dependence distance tiling, the unique set oriented partitioning, and three region partitioning methods. In the proposed method, we eliminate anti dependences from the nested loop by variable renaming. After variable renaming, we present algorithm to select one or more appropriate lines among given four lines such as LMLH, RMLH, LMLT and RMLT. If only one line is selected, the method divides the iteration space into two parallel regions by the selected line. Otherwise, we present another algorithm to find a serial region. The selected lines divide the iteration space into two parallel regions as large as possible and one or less serial region as small as possible. Our proposed method gives much better speedup and extracts more parallelism than other existing three region partitioning methods.

• Key Words : Parallel Compiler; Three Region Partition; Irregular Dependences; Nested Loop; Variable Renaming

Abstract 본 논문은 비규칙 종속성을 가진 내포된 루프의 수행 속도를 향상시키기 위해서 Extended Three Region Partitioning Method 라는 효과적인 루프 분할 방법에 대해서 연구하였다. 본 논문에서 제안된 루프 분할 방법은 변수 재명명에 의해서 역종속성을 가진 내포된 루프를 제거한 후 네 개의 선중에 하나 혹은 그 이상의 적절한 선을 선택하는 알고리즘을 개발한다. 한 개의 선이 선택되면 선택된 선에 의해서 전체 영역은 두 개의 병렬지역으로 분할된다. 한 개 이상의 선이 선택되면 그 선들에 의해서 하나의 순차지역과 두 개의 병렬지역으로 분할한다. 제안된 분할 방법은 기존의 분할 방법보다 성능이 우수함을 성능 분석에서 보여준다.

• Key Words : 컴파일러, 세지역 분할 방법, 비규칙 종속성, 네포된 루프, 변수 재명명

1. Introduction

Computationally expensive programs spend most of their time in the execution of DO-loops [1][2]. Partitioning of loops is a very important optimization issue and requires the efficient and exact data dependence analysis [3][4]. A lot of work has been done in parallelizing nested loops with uniform dependences, from dependence analysis to loop transformation [5-9].

*교신저자 : 정삼진(sjjeong@bu.ac.kr)
접수일 2015년 3월 16일 수정일 2015년 4월 29일 게재확정일 2015년 6월 20일

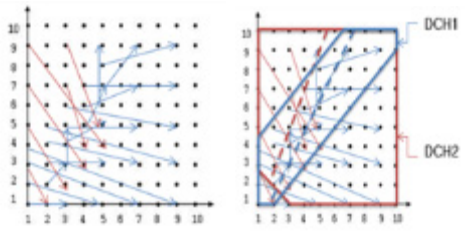
Example 1.

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

Some parallelization techniques, based on Convex Hull theory [10] which has been proved to have enough information to handle irregular dependences, are minimum dependence distance tiling method [11], the unique set oriented partitioning method [12] and three region partitioning method [13], [14].

This paper will focus on parallelization of flow and anti dependence loops with irregular dependences.

Example 1 illustrates an irregular dependence loop. [Fig. 1(a)] shows the dependence loop. [Fig. 1(a)] shows the dependence patterns of Example 1 in the iteration space.



(a) (b)

[Fig. 1] (a) Iteration spaces with Irregular dependencies (b) CDCH of Example

2. Program Model and Dependence Analysis

We consider doubly nested loop program of the form shown in [Fig. 2]. For the given loop, l1(l2) and u1(u2) indicate the lower and upper bounds respectively, and should be known at compile time. To characterize the coupled array subscripts, the array subscripts, f1(I,J), f2(I,J), f3(I,J), and f4(I,J), are linear functions of the loop index variables.

```
do I = l1, u1
  do J = l2, u2
S1 : A(f1(I, J), f2(I, J)) =  .  .  .
S2 :        .  .  .  = A(f3(I, J), f4(I, J))
  enddo
enddo
```

[Fig. 2] A doubly nested loop model

In our program model statement S1 defines elements of array A and statement S2 uses them. Dependence exists between S1 and S2 whenever both refer to the same element of array A. If the element defined by S1is used by S2 in a subsequent iteration, then flow dependence exists between S1 and S2. On the other hand, if the element used in S2 is defined by S1 at a later iteration, this dependence is called anti dependence.

The doubly nested loop carries cross-iteration dependences if there exist two distinct iteration vectors (i1, j1) and (i2, j2), such that f1(i1, j1) = f3(i2, j2) and f2(i1, j1) = f4(i2, j2). More specifically, the nested loop in [Fig. 2] carries cross iteration dependences if and only if there exist four integers (i1, j1, i2, j2) 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 [12]. Finding the general solution of a system of Diophantine equations is time-consuming when the level of the nested loop is large [12]. However, in real application programs, the level of nested loops is usually small. For a doubly nested loop as shown above, this can be solved in about 300 μsec on a SPARC-2 workstation [10].

$$f1(i1, j1) = f3(i2, j2) \text{ and } f2(i1, j1) = f4(i2, j2) \quad (1)$$

$$l1 \leq i1, i2 \leq u1 \text{ and } l2 \leq j1, j2 \leq u2 \quad (2)$$

There are four unknown variables in our program model. Here we consider only those cases for which we can express the general solution in terms of two integer variables. So we have integer solutions

(i1,j1,i2,j2) which are functions of x and y. They can be written as

$$(i1, j1, i2, j2) = (g1(x, y), g2(x, y), g3(x, y), g4(x, y)) \quad (3)$$

Here gi are functions with integer coefficients.

Let R denote the set of real numbers. The two free variables of a general solution (3), x and y, form a 2-dimensional space, namely solution space Γ given by $\Gamma = \{(x, y) | x, y \in R\}$.

Every integer point (x, y) in the solution space corresponds to two vectors, (g1(x, y), g2(x, y))and (g3(x, y), g4(x, y)), which possibly forms a dependence vector. Any integer point (x, y) in this solution space causes dependence between statements S1 and S2, provided the system of inequalities given by (2) is satisfied. In terms of the general solution, the system of inequalities can be given as

$$l1 \le g1(x,y) \le u1,\ l2 \le g2(x,y) \le u2 \quad (4)$$
$$l1 \le g3(x, y) \le u1,\ l2 \le g4(x, y) \le u2$$

These inequalities limit the solution space, and form a convex polyhedron, which can also be termed as Dependence Convex Hull (DCH) [12].

From (1) and (2), two sets of inequalities can be written as

$$l1 \le g3(i1,j1) \le u1 \text{ and } l2 \le g4(i1,j1) \le u2 \quad (5)$$
$$l1 \le i1 \le u1 \text{ and } l2 \le j1 \le u2$$
$$l1 \le g1(i2,j2) \le u1 \text{ and } l2 \le g2(i2,j2) \le u2 \quad (6)$$
$$l1 \le i2 \le u1 \text{ and } l2 \le j2 \le u2$$

We can write these dependence distance functions in a general form as

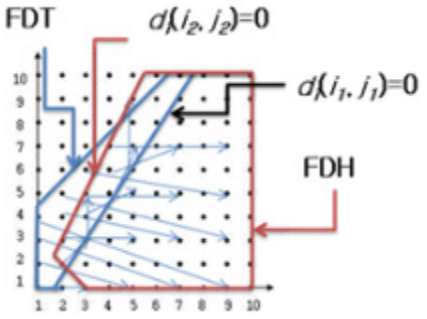
$$d(i1, j1) = (di(i1, j1), dj(i1, j1)), \quad (7)$$
$$d(i2, j2) = (di(i2, j2), dj(i2, j2))$$
$$di(i1, j1)=p1*i1+q1*j1+r1,\ dj(i1, j1)=p2*i1+q2*j1+r2$$
$$di(i2, j2)=p3*i2+q3*j2+r3,\ di(i2, j2) =p4*i2+q4*j2+r4$$

where pi, qi, and ri are real values and i1, j1, i2, and j2are integer variables of the iteration space.

And, (5) and (6) form DCHs denoted by DCH1 and DCH2, respectively [12]. The union of DCH1 and DCH2 is called Complete DCH (CDCH), and all dependences lie within the CDCH. [Fig. 1(b)] shows the CDCH of Example 1.

3. Extended Three Region Partitioning Method

In this section, we present an extended three region partitioning method to partition doubly nested loops with irregular dependence sets. If the line di(i, j) = 0 passes through the CDCH, then it divides a DCH into DCH1 and DCH2 as shown in [Fig. 1(b)].



[Fig. 3] FDT and FDH of Example 1

We defined FDT and FDH as follows[16].

Definition 1 : If line di(i1,j1) = 0 intersects DCH1, the flow dependence tail set of the DCH1 is called as FDT.

Definition 2 : If line di(i2,j2)=0 intersects DCH2, the flow dependence head set of the DCH2 is called as FDH.

[Fig. 3] shows the flow dependence tail set (FDT) of DCH1 and the flow dependence head set (FDH) of DCH2 of the loop in Example 1 after variable renaming.

We also define the line LMLT (Left Most Line) and the line RMLT (Right Most Line) in FDT, and the line LMLH (Left Most Line) and the line RMLH (Right

Most Line) in FDH as follows.

Definition 3 : The line that can be formed by the two left most extreme points in FDT is called the LMLT (dli(i1, j1) = 0). And the line by the two left most extreme points in FDT is called RMLT (dri(i1, j1)=0).

Definition 4 : The line that can be formed by the two left most extreme points in FDH is called the LMLH (dli(i2,j2)=0). And the line by the two left most extreme points in FDHs called RMLH (dri(i2,j2)=0).

Property 1 : Suppose line di(i, j) = p*i + q*j + r passes through CDCH.

If q > 0, FDT(FDH) is on the side of di(i1, j1)≥ 0 (di(i2, j2)≥ 0), otherwise, FDT(FDH) is on the side of di(i1, j1)≤ 0 (di(i2, j2)≤ 0).

In the given algorithm Region_Partition[15], we select one or more appropriate lines among four lines such as the line LMLT and the line RMLT in FDT, and the line LMLH and the line RMLH in FDH. The line di(i1, j1) = 0 and the line di(i2, j2) = 0 are included in these four lines. The line di(i1, j1) = 0 is one of two lines LMLT and RMLT in FDT, and the line di(i2, j2) = 0 is one of two lines LMLH and RMLH in FDH. After selecting one or more appropriate lines in two-dimensional solution space, Algorithm Region_Partition executes Serial_Region, which is the algorithm of finding a serial region from three lines as shown in [Fig. 4], and is similar to the algorithm presented in [16].

Algorithm Finding_Serial_Region

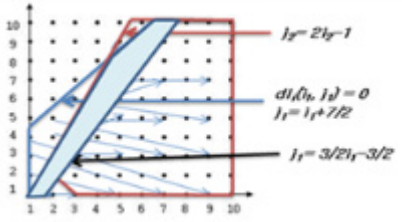
Input: Three Lines

Output: An Serial_Region;

```
                struct node {
                  float (x, y);
                  int zoom;
                  struct node *next;
                  struct node *prev; };
                max = 9999999;
begin
Build the initial serial region ring which is composed of five nodes:
    (x1,y1)=(max,max);
    (x2,y2)=(max,-max);
    (x3,y3)=(-max,-max);
    (x4,y4)=(-max,max);
    (x5,y5)=(-max,max);
  while (the input list is not empty)
    Pop a half space from the list, named HS;
    Scan the ring;
      { Determine the zoomvalueforeachnode;}
      if ((x,y)∈HS) then
        zoom=0;
      else
        zoom=1;
 I    if (the zoom is different from previous node)
then
       { Compute the intersection point
         and give it zoom=0;
         Insert it into the ring between the
         current node and the previous node };
      endif
    Scan the ring again;
      { Remove the nodes with zoom=1};
    if (the ring is empty) STOP;
  end while
end Finding_Serial_Region
```

[Fig. 4] Algorithm of finding a serial region.

By using an example given in Example 1, we can get three lines from algorithm Region_Partition. The first one is di(i2, j2) = 0, the second one is dli(i1, j1) = 0, and the third one is di(i1, j1) = 0.



[Fig. 5] Regions of the loop partitioned by the extended region

[Fig. 3] shows case that the line di(i1, j1) = 0 is on the left side of the line di(i2, j2) = 0, and q1>0 and q3<0.

Algorithm Region_Partition selects three lines di(i2, j2) = 0 and dli(i1, j1) = 0 (=LMLH), and di(i1, j1) = 0 (=RMLT) among four given. These three selected lines are the boundaries of three loops. After selecting three appropriate lines, the algorithm executes algorithm Serial_Region.

[Fig. 5] shows regions of the loop partitioned by our proposed technique in Example 1. From the algorithm Serial_Region, two lines (j = 2i2 - 1 and j = i1+ 7/2) can be the upper bound in the serial region AREA2 and lower bound in AREA3. The third line j = 2i2- 1 is upper bound in AREA1 and lower bound in AREA2. In this case, the iteration space is divided into two parallel regions, AREA1 and AREA3, and one serial region, AREA2 by the three selected lines j = 2i2 - 1, j = i1+ 7/2 and j = 3/2i1 - 3/2 2 as shown in [Fig. 5]. The execution order is AREA3 → AREA2 → AREA1.

4. Performance Analysis

Theoretical speedup for performance analysis can be computed as follows. 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:

$$\text{Speedup} = (Ni * Nj)/(Np + Ns)$$

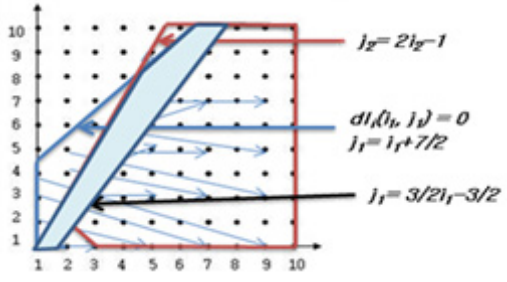
where Ni,Nj are the size of loop i, j, respectively

By using an example given in Example 1, we compare the performance of our proposed method with that of related works. The improved three region partitioning [13] can divide the iteration space into two parallel regions, AREA1 and AREA3, and one serial region, AREA2, by line di(i2, j2) = 0 (j2 = 2i2 - 1) and line di(i1, j1) = 0 (j1 = 3/2i1 - 3/2) as shown in [Fig. 6]. The speedup can be computed as (10*10)/(2+17) = 5.26.

The three region partitioning [10] is divides the iteration space into one parallel region, AREA1, and one serial region by line di(i1, j1) = 0 (j1 = 3/2i1 - 3/2) as shown in [Fig. 6]. The speedup can be computed as (10*10)/(1+69) = 1.43.

Applying the proposed method to this loop is the case that FDT overlaps with FDH and line di(i1, j1) = 0 does not intersect line di(i2, j2) = 0. After variable renaming, there remains only flow dependence as shown by the partitioning in [Fig. 3]. From the algorithm Serial_Region, two lines (j = 2i2 - 1 and j = i1+ 7/2) are the upper bound in the serial region AREA2 and lower bound in AREA3. The third line j = 2i2- 1 is upper bound in AREA1 and lower bound in AREA2. In this case, the iteration space is divided into two parallel regions, AREA1 and AREA3, and one serial region, AREA2 by the three selected lines j = 2i2 - 1, j = i1+ 7/2 and j = 3/2i1 - 3/2 2 as shown in [Fig. 5]. The execution order is AREA3 → AREA2 → AREA1. The speedup for this method is (10*10)/(2+14) = 6.25.

In the above comparisons, our proposed partitioning method exploits more parallelism than the other related methods.



[Fig. 6] Regions of the loop partitioned by the improved three region partitioning method in Example 1

5. Conclusions

In this paper, we studied the problem of

transforming nested loops with irregular dependences, and proposed efficient methods such as extended three region partitioning method to maximize parallelism.

In our proposed method, the Extended Three Region Partitioning Method, we select one or more appropriate lines among four lines such as LMLH, RMLH, LMLT and RMLT, as given in Definition 3 and 4. One or more selected lines divide the iteration space into two parallel regions and/or less than one serial region.

In comparison with some previous partitioning methods, such as minimum dependence distance tiling, unique sets oriented partitioning and three region partitioning, our technique gives much better speedup and extracts more parallelism than other existing methods.

ACKNOWLEDGMENTS

본 논문은 2015년도 백석대학교 대학연구비에 의해서 수행된 것임.

REFERENCES

[1] D. Bilar, E. Filiol, “On self-reproducing computer programs,” in Journal in Computer Virology, vol. 9, no. 1, pp. 9-87, Feb. 2009

[2] S. P. Midkiff, “Automatic Parallelization : An Overview of Fundamental Compiler Techniques,” in Synthesis Lectures on Computer Architecture, vol. 7, no.1, pp. 1 -169, 2012.

[3] U. Banerjee, Dependence Analysis for Supercomputing, Kluwer Academic, Norwell, Mass., 1988.

[4] W. Pugh, "the Omega test: A fast and practical integer programming algorithm for dependence analysis," in Proc. Supercomputing'91, Nov. 1991.

[5] M. Wolfe and C. W. Tseng, "The power test for data dependence," IEEE Trans. Parallel and Distributed Systems, vol. 3, no. 5, pp. 591~601, Sep. 1992.

[6] U. Banerjee, Speedup of Ordinary Programs, Ph.D. thesis, University of Illinois at Urbana-Champaign, Oct. 1979.

[7] M. E. Wolfe and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," IEEE transactions on Parallel and Distributed Systems, vol. 2, pp. 452~471, Oct 1991.

[8] D. Maydan, J. Hennessy, and M. Lam, "Efficient and exact data dependence analysis," in Proc. ACM SIGPLAN '91 Conf. on programming Language Design and Implementation, Toronto, June 1991.

[9] S. Chatterjee, "Compiling nested data-parallel programs for shared-memory multiprocessors," ACM Transactions on Programming Languages and Systems, vol. 15, no. 3, pp. 400~462, July 1993.

[10] T. Tzen and L. Ni, "Dependence uniformization: A loop parallelization technique," IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 5, pp. 547-558, 1993.

[11] S. Punyamurtula and V. Chaudhary, "Minimum dependence distance tiling of nested loops with non-uniform dependences," in Proc. Symp. Parallel and Distributed Processing, pp. 74-81, 1994.

[12] J. Ju and V. Chaudhary, "Unique sets oriented Partitioning of nested loops with non-uniform dependences," in Proc. Int. Conf. Parallel Processing, vol. 3, pp. 45-52, 1996.

[13] C. K. Cho and M. H. Lee, "A Loop Parallization Method for Nested Loops with Non-uniform Dependences", in Proceedings of the International Conference on Parallel and Distributed Systems, pp. 314-321, 1997.

[14] D. L Pean and C. Chen, “An Optimized Three Region Partitioning Technique to Maximize Parallelism of Nested Loops with Non-uniform Dependences," Journal of Information Science and Engineering, vol. 17, pp. 463-487, 2001.

[15] S. J. Jeong, “Maximizing Parallelism for Nested Loops with Non-uniform Dependences”, pp. 213-222, ICCSA 2004.

[16] T. Tzen and L. Ni, "Dependence uniformization: A

loop parallelization technique," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 5, pp. 547-558, 1993.

저자소개

정 삼 진(Sam-Jin Jeong) **[정회원]**

- 1987년 2월 : 인디애나대학교 컴퓨터학과 (컴퓨터학 석사)
- 2010년 8월 : 충남대학교 컴퓨터과학과 (이학박사)
- 2003년 3월 ~ 2006년 2월 : 한국연구소 책임연구원
- 1987년 3월 ~ 현재 : 백석대학교 정보통신학부 교수

<관심분야> : 병렬 컴파일러, 프로그래밍 언어, 모바일 컴퓨팅