Parallelization for Multi-dimensional Loops with Non-uniform Dependences

Sam-Jin Jeong

***Abstract*—This paper presents two enhanced partitioning algorithms in order to improve parallelization of multi-dimensional loops with flow and anti-dependences Using distance of the first dependence, we show the first general enhanced algorithm of single-loops with both flow and anti-dependences. Using the first algorithm, loop interchanging method, and cycle shrinking method, we present parallelization of nested loops with simple subscripts. Using the second algorithm, we also present parallelization to multi-dimensional loops with both flow and anti-dependences. Our two presented algorithms show enhanced loop parallelization of multi-dimensional loops with both dependences. We will improve our proposed algorithms for multi-dimensional space with multiple dependences.**

***Index Terms*—Parallelizing Compiler, Multi-dimensional Loops, Multiple Dependences, Loop Transformation, Non-uniform Dependence**

Ⅰ. INTRODUCTION

A parallelizing compiler is the good solution for parallelization for software engineers. Tasking serial programs, it gives parallelization opportunities, executes source code transformations results[1]. A lot of actual software spends many times to the execution of DO loops[2]. In computationally expensive programs, concentrating on the parallelization in a loop is an enhanced approach for exploiting parallelization[3]. The loop transformation requires accurate data dependence analysis[4,5]. Accurate dependency analysis helps identify the dependent / independent iteration of the loop. In order to achieve maximum parallelism, the appropriate dependence analysis is important. We review several data dependence tests about the dependency of one-dimensional loops[6,7]. First of all, we generally applied GCD test because it is simple. Second, the separability test is applied. And it is possible to obtain additional information through the test. When we are considering approaches to single-loop, we can exanimate two splitting methods such as fixed splitting method with minimum distance and variable splitting methods[8].

Manuscript received June 15, 2017, revised August 22, 2017

S. J. Jeong is with Division of Information and Communication Engineering, Baekseok University, #76, Munam-ro, Cheonan, Chungnam, 330-704, Korea (e-mail: sjjeong@bu.ac.kr)

But, some parallelization unexploited is leaved. Chapter two will introduce some splitting methods such as Polychronopoulos' technique and splitting technique by thresholds. In chapter three, we propose two enhanced loop transformation algorithms to exploit loop parallelization of multi-dimensional loops with non-uniform dependences. The conclusion is made in chapter four.

Ⅱ. RELATED WORKS

DO $I = p, q$
  $A(a_1 * I + a_2) = \cdot\cdot$
    $\cdot\cdot = A(b_1 * I + b_2)$
END

Fig. 1. Single-loop model

Figure 1 shows a general form of single-loop. In single-loop, there are two variables and those loop variables are components of one dimensional array for data-dependence. We introduce two splitting methods of single-loops now. We can present some parallelization available in a single-loop given in Figure 1. We can classify four cases for integer a1 and integer b1, which are coefficients of the index variable I given by equation (1).

Case I: $a1 = b1 = 0$;
Case II: $a1 = 0;\ b1 \neq 0$ or $a1 \neq 0;\ b1 = 0$;
Case III: $a1 = b1 \neq 0$;
Case IV: $a1 \neq 0;\ b1 \neq 0;\ a1 \neq b1$; (1)

2. A Loop partitioning technique using thresholds

Loop partitioning technique using threshold was first published by Allen and Kennedy. They proposed two loop splitting methods, that are loop partitioning technique using cross threshold and loop partitioning technique using constant threshold. Loop partitioning technique using cross threshold is applied the case IV of equation (1). Loop splitting method using constant threshold is applied the case III of equation (1).

2. B Polychronopoulos' loop partitioning technique

When a1 * b1 => 0, we can make use of any parallelization for the case IV of equation (1). We will consider three cases whether it exists both dependences. Let regard (i, j) as an integer to equation (1). If the first distance of i in equation (2) is positive, there is flow dependence. If the second distance of j in equation (3) is positive, there is anti-dependence. If (x, x) has a solution of equation (1), equation d(x) = 0 and da(x) = 0, and there are both dependences after and before i = x. Equation A(a1i + a2) cannot be expended before d(i), and it means which d(i) can perform in parallel for each value of i.

d(i) = j – i, d(i) = D(i)/b1; where D(i) = (a2 - b1) i + (a2 - b1) (2)
da(j) = i – j, da(j) = Da(j)/a1; where Da(j) = (b1 - a1) j + (b2 - a2) (3)

Ⅲ. Parallelization for Nested Loops

Using distance of the first dependence, we offer general enhanced algorithm of multi-dimensional loops with both flow and anti-dependences as follows.

A Transformation of Single-loops

Procedure LoopSplit presents the parallelization of single-loops[9]. The Procedure LoopSplit2 shows how to split single-loops.

Procedure LoopSplit2
BEGIN
Step 1: Data dependence testing.
Step 2: Data transformation
Step 3: The case satisfying IV of equation (1)
Step 4: The case with one dependence
Step 5: Call LoopSplit(a);
Step 6: The case with both dependences
Step 7: Call LoopSplitb(b);
Step 8: Call LoopSplitc(c);
Step 9: Merge two splitted blocks;
END LoopSplit2

In step 5-6, if there exists both dependances, Procedure LoopSplit divides the loop two parts, and transforms each of parts.

B. Transformation of Loops with Multiple Dependences

In the section 3.1, we considered only the case with dependence in single-loops. In this section, by extending the transformation method of the case with dependence in single-loops, a transformation of loops with multiple dependences is presented. If we suppose there are *m* non-uniform dependences in a single-loop, then Procedure MultiSplit shows the algorithm to exploit parallelization in single-loops with multiple dependences.

Procedure MultiSplit (l, p, s_d[], α[], β[])
/* parallelization of single-loops with multiple dependences */
BEGIN
/* Find the first one in the *i*th block.
St[k]: the first source iteration in any block of each of *m* dependences.
$d_k(i)$: the distance at any iteration *I* for each of *m* dependences.
S_d[k]: the difference between two adjacent source iterations for each of *m* dependences.
α[k], β[k]: the iteration and distance of the first source for each of *m* dependences computed by the separability test, respectively */
Step 1: i = 1; St[1] = l;
Sr[k] = α[k] and d_k(Sr[k]) = β[k] for $1 \le K \le m$;
Step 2: St[i+1] = min {Sr[k] + d_k(Sr[k]} for $1 \le K \le m$;
If St[i+1] $\ge p$, then {St[i+1] = p + 1; goto Step 5};
Step 3: Sr[k] = St[I+1] + q[k] for $1 \le K \le m$,
Where $0 \le$ q[k] $\le S_d$[k] and q[k] = (α[k] – St[k+1]) mod S_d[k];
Step 4: Compute d_k[k](Sr[k]) for $1 \le K \le m$;
i = i + 1; goto Step 2;
Step 5: /* split the loop into blocks with variable sizes, St[i+1] – St[i]. */
END MultiSplit

C. Transformation of Nested Loops with Simple Subscripts

In previous sections, we proposed a generalized and optimal method for single-loops only. This section discusses the extension of the first method, in order that it can be applied to present parallelization of nested loops with simple subscripts. However, it is difficult to apply this method to nested loops with coupled subscripts. If we consider nested loops with simple subscripts as given in Figure 2, we can present an enhanced method for these loops by extending the first method, based on cycle shrinking [8,10] and loop interchange[11].

```
DO I1 = p1, q1
  DO I2 = p2, q2
   ···
    DO In = pn, qn
      A(f1(I1), ··· , fn(In)) = ···
                ··· = A(g1(I1), ··· , gn(In))
    END
   ···
  END
END
```

Fig. 2. A type of nested loop with simple subscripts

Since our loop model given in Figure 2 is the type of nested loop with simple subscript, here the data dependence is considered separately for each individual loop in the nest. Each loop of this nested loop transfers cross-iteration dependences if there is two integers (i, j) satisfying inequalities (5) and Diophantine equations (4).

$$f_k(I_k) = g_k(I_k) \rightarrow a_{k1}I_k + a_{k2} = b_{k1}I_k + b_{k2} \text{ for } 1 \le k \le n \quad (4)$$
$$l_k \le i \le u_k \text{ and } l_k \le j \le u_k \quad (5)$$

If each component of the distance vector (6) is positive, there is a flow dependence, and if value of $d_{ak}(j)$, equation (7), is positive, there is an anti-dependence. And, if equation (4) has a solution, $d_k(x) = 0$ and $d_{ak}(x) = 0$, and there are both dependences.

$$d_k(i) = j - i = D_k(i)/b_{k1},\ D_k(i) = (a_{k1} - b_{k1})i + (a_{k2} - b_{k2}) \quad (6)$$
$$d_k(j) = j - i = D_{ak}(j)/a_{k1},\ D_{ak}(j) = (b_{k1} - a_{k1})i + (b_{k2} - a_{k2})$$
(7)

We can briefly present our proposed method as follows. First, using the procedures in section 3.1, the number of blocks which can be splitted form iteration space is computed for each loop in the nest starting with outermost loop. Next, *k*th loop which has minimum number of blocks in the nested loop, and the kth and the outermost loops (L_k and L_1) are interchanged for maximizing parallelization available in the loop 10. Then the outermost loop interchanged (old L_k) is blocked, and all loop nested inside the outermost loop are transformed to DOALL's. Even if only a loop in the nest does not have the dependence, all loops can be transformed to DOALL's. We can consider the proposed method in two cases: one is that one type of dependence exists in the loop and the other is that both flow and anti-dependence exist in the loop. Here, the number of blocks B_k for each loop in the nest can be computed by Procedure Compute_NB. When there exists a loop-independent dependence in the kth loop.

Procedure Compute_NB
/* Computation of the number of blocks for each loop in the nested loop */
BEGIN
k = 1 ;
While $k \le n$ Do
If ($a_{k1} = b_{k1} = 0$) then {$B_k = 1$; $F_k = 0$} ;
Orif ($a_{k1} = 0$, $b_{k1} \neq 0$ or $a_{k1} \neq 0$, $b_k = 0$) then {
If ($l_k \le i \le u_k$ where $i = (b_{k2} - a_{k2})/a_{k1}$ (if $a_{k1} \neq 0$) or
$(a_{k2} - b_{k2})/a_{k1}$ (if $b_{k1} \neq 0$))
then $B_k = 3$ else $B_k = 2$; $F_k = 1$};
Orif ($a_{k1} = b_{k1} \neq 0$) then{
If($a_{k2} = b_{k2}$) then $B_k = 1$
else $B_k = \lceil (u_k - l_k)/(a_{k2} - b_{k2})/b_{k1} \rceil$ (if $(a_{k2} - b_{k2})/b_{k1} > 0$) or
$\lceil (u_k - l_k)/(b_{k2} - a_{k2})/a_{k1} \rceil$ (if $(b_{k2} - a_{k2})/a_{k1} > 0$); $F_k = 0$};
Orif ($a_{k1} * b_{k1} < 0$) then { $B_k = 2$; $F_k = 0$};
Orif ($\exists$ only a flow or anti-dependence in the kth loop) then {
Compute B_k by step 1-4 in Procedure LoopSplit; $F_k = 0$ }
else {Compute B_k by step 5-7 in Procedure LoopSplit2; $F_k = 1$}
k = k+1;
Endwhile
END Compute_NB

First, in case that one type of dependence exists in each loop of a nested loop with simple subscripts, Procedure LoopSplit_3 can transform a nested loop into partial parallel loops. As an example, let's consider the loop shown in Figure 6. There is one type of dependence exists in each loop of this nested loop. The number of blocks of L_2 is 4 and one of L_3 is 3. Hence, L_3 is interchanged with the outermost loop L_1 for maximizing parallelization.

Procedure LoopSplit_3
/* Transformation of nested loops with simple subscripts */
BEGIN
Step 1: Compute the number of blocks B_k by Procedure Compute_NB for each loop L_k such that the dependence distance $d_k > 1$ for $1 \le k \le n$;
Step 2: Find L_1 such that $B_i = \min(B_k)$ for 1 <= k <= n;
Step 3: If i > 1, then interchange the first loop L_1 with the ith loop L_i ;
Step 4: If the dependence distance of the outermost loop is constant then split the outermost loop into partial parallel loops by the reduction factor λ (assuming the outermost loop is L_i, $\lambda = \lceil (a_{i2} - b_{i2})/b_{i1} \rceil$ (if $d_i(i) > 0$) or $\lceil (b_{i2} - a_{i2})/a_{i1} \rceil$ (if $d_{ai}(j) > 0$))
else split the outermost loop by Procedure LoopSplit_1;
Transform all loops nested inside the outermost loop, i.e., from the second loop to the nth loop, to DOALL's;
End LoopSplit_3

D. Transformation of Multi-dimensional Loops with Non-uniform Dependences.

Now, let's consider a case which there exist both dependences in the loop. A loop-independent dependence in a single-loop does not cause any problem for parallelizing a loop. Namely, L_{i+1} may cause to take place a cross-iteration dependence at the iteration where a loop-independent dependence exists in () in the nest. In case of the loop in Figure 3, loop-independent dependence exist in L_1 at $I_1 = 5$and in L_3 at I_3=3 as shown in Figure 5 shows the unrolled version of the loop in Figure 3, when $I_1 = 5$, and there exist flow (anti-) dependences represented as arrows. However, if we splitL_2, the inner loop of L_1, as shown in Figure 5, we can remove this dependences. Then, when the distance of the inner loop is zero, we cannot split the nested loop at this iteration. And, as mentioned above, we can interchange the outermost loop with the loop which has the minimum number of blocks for maximizing parallelization.

```
DO I1 = 1, l
   DO I2 = 1, m
      DO I3 = 1, n
         A(2*I1, 3*I2, 2*I3) = . .
                  . . = A(I1+5, I2+11, I3+3)
      END
   END
END
```

Fig. 3. An example of nested loop with both flow and anti-dependences

I_1	A(2*I_1)	A(I_1+5)I_3
1	A(2)	A(6)
2	A(4)	A(7)
3	A(6)	A(8)
4	A(8)	A(9)
5	A(10)	A(10)
6	A(12)	A(11)
7	A(14)	A(12)
8	A(16)	A(13)
9	A(18)	A(14)
10	A(20)	A(15)

I_3	A(2 I_3)	A(I_3+3)
1	A(2)	A(4)
2	A(4)	A(5)
3	A(6)	A(6)
4	A(8)	A(7)
5	A(10)	A(8)
6	A(12)	A(9)
7	A(14)	A(10)
8	A(16)	A(11)
9	A(18)	A(12)
10	A(20)	A(13)

Fig. 4. The unrolled versions of L1 and L3 of the loop in Figure 3

I_2	A(2 I_1, 3 I_2,2 I_3)I_3	A(I_1+5, , I_2+11, I_3+3)
1	(10, 3, *)	(10, 12, *)
2	(10, 6, *)	(10, 13, *)
3	(10, 9, *)	(10, 14, *)
4	(10, 12, *)	(10, 15, *)
5	(10, 15, *)	(10, 16, *)
6	(10, 18, *)	(10, 17, *)
7	(10, 21, *)	(10, 18, *)
8	(10, 24, *)	(10, 19, *)
9	(10, 27, *)	(10, 20, *)
10	(10, 30, *)	(10, 21, *)

(note) * : any integer in $1 \le * \le 10$

Fig. 5. The unrolled versions of the loop in Figure 3 when I1 = 5

Procedure LoopSplit_4 generalize how to partition the general loop to partial parallel loops as shown in Figure 3.

Procedure LoopSplit _4
/* Transformation of nested loops with simple subscripts to parallel loops */
BEGIN
Step 1: /* To test for data dependence and computer the number of blocks B_k for each loop in the nest */
k=1 ;
While $k \le n$ Do
If GCD is not true, then transforms nested loops to parallel loops;
If separability test is not true, then transforms nested loops to parallel loops;
Compute B_k and F_k by Procedure Compute_NB ;
k = k + 1 ;

Endwhile
Step 2: /* To find the loop L_i which will be interchanged with the outermost loop L_1. */
i =0 ;
Find L_i such that $B_i = \min(B_k)$ for $1 \le k \le n$ & $B_k > 1$;
If i = 0 then transforms nested loops to parallel loops; /* The absence of L_i in the loop. */
If $F_i = 0$ then go to Step 3; /* The absence of a loop-independent dependence (LID) in the loop. */
j = 0 ;
Find L_j such that $B_j = \min(B_k)$ for $1 \le k \le n$ & $k \ne I$ & $B_k > 1$; /* To find the loop L_j such that B_j is the smallest except B_i in the nest */
$B_i = B_i + B_j - 1$;
m = 0 ;
Find L_m such that $B_m = \min(B_k)$ for $1 \le k \le n$ & $k \ne I$ & $B_k > 1$ &$F_k = 0$; /* To find the loop L_m except L_i in which LID does not exist. */
If m = 0 then go to Step 3 ; /* The absence of L_m. */
Find L_i such that $B_i = \min(B_i, B_m)$ /* To find again the loop L_i which will be interchanged with the outermost loop L_1. */
If $F_i = 1$ & $j \ne 2$ then interchange L_2 with L_j ;
Step 3: If i> 1 then interchange L_1 with L_i ;
Step 4: If LID does not exist in the outermost loop (i.e., when L_i is the outermost loop, $F_i = 0$) then the same as Step 4 in Procedure LoopSplit_3
Step 5: else { split the outermost loop by the same as Step 4 in Procedure LoopSplit2 ;
for all splitted blocks of the outermost loop except the block where LID exists, transforms all loops nested inside the outermost loop to DOALL's ;
for the block where LID exists,
If j = 0 then leave all loops nested inside the outermost loop as they are original loops
else { split the second loop by the same as Step 4 ;
transform all loops nested inside the second loop to DOALL's } }
;
Stop ;
/* The results of the loop in Figure 4.4 transformed by Step 4-5

```
(1) The absence of LID in Li                          (2) The presence
    of LID at x in Li
    DO Ii' = li, ui, λi                                DO Ii' = li, x-1,
λi
      DOALL I1 = Ii', min(ui, Ii'+λi-1)                  DOALL Ii =
    Ii', min(x-1, Ii'+λi-1)
        DOALL I2 = l2, u2                               DOALL
          . . .                                          . . .
          DOALL I1 = l1, u1                            DO Ii' = x, x
            DOALL Ii+1 = li+1, ui+1                      DO I2' = l2, u2,
    λ2
              . . .                                       DOALL I2
    = I2', min(u2, Ii'+λ2-1)
                DOALL In = ln, un                          . . .
                  . . .                                DO Ii' = x+1, ui,
    ii
                ENDDO                                    DOALL Ii = Ii',
    min(ui, Ii'+λi+1)
                  . . .                                   DOALL
                                                           . . .
```

END **LoopSplit_4**

Figure 6 shows the result of the loop in Figure 3 transformed by Procedure LoopSplit_4. In step 2, B_is are computed as $B_1 = 5$, $B_2 = 5$ and $B_3 = 4$. The third loop can be splitted as the smallest number of blocks of them. However, since there is a loop-independent dependence in $L3$, $L2$ without a loop-independent dependence is selected as the outermost loop ($B_2 = 5 < B_3 = B_3 + B_1 = 9$).
. . .

```
DOALL I2 = 1, 3                    DOALL I2 = 5, 6
  DOALL I1 = 1, 10                   DOALL I1 = 1, 10
    DOALL I3 = 1, 3                    DOALL I3 = 5, 6
      S1 ;                               . . .
      S2 ;                           DOALL I2 = 7, 9
    ENDDO                              DOALL I1 = 1, 10
  ENDDO                                  DOALL I3 = 1, 10
ENDDO                                      . . .
DOALL I2 = 4, 4                      DOALL I2 = 10, 10
  DOALL I1 = 1, 10                     DOALL I1 = 1, 10
    DOALL I3 = 1, 10                     DOALL I3 = 1, 10
      . . .                                . . .
```

Fig. 6. The result of the loop in Figure 3 transformed by Procedure LoopSplit_4

IV. Conclusion

In this research, we have presented the parallelization of multi-dimensional loops with both dependences and nested loops with simple subscripts in order to improve parallelization. For single-loops, we introduce two partitioning techniques such as loop partitioning technique by thresholds and Polychronopoulos' loop partitioning technique. But, some parallelization unexploited is leaved, and the second one has some dependence restrictions. Therefore, we proposed two generalized enhanced algorithms in order to improve parallelization. Using distance of the first dependence, we show the first enhanced algorithm for single-loops with both flow and anti-dependences. Using the first algorithm, loop interchanging method, and cycle shrinking method, we present the second generalized algorithm for parallelization of nested loops with simple subscripts. Using the second algorithm, we also present parallelization for multi-dimensional loops with both dependences. Our two presented algorithms show enhanced loop parallelization of multi-dimensional loops with both dependences. We will improve our proposed algorithms for multi-dimensional space with multiple dependences.

Acknowledgment

This WORK is supported by 2017 Research fund from Baekseok University.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," *Phil. Trans. Roy. Soc.* London, vol. A247, pp. 529-551, Apr. 1955.
[2] J. Clerk Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68-73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in *Magnetism*, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.
[4] T. L. Gilbert, *Formulation, Foundations and Applications of the Phenomenological Theory of Ferromagnetism*, Ph.D. dissertation, Illinois Inst. Tech., Chicago, IL, 1956, unpublished.
[5] D. P. Arnold, "Review of microscale magnetic power generation," submitted for publication.
[6] S. O. Demokritov and V. E. Demidov, "Micro-Brillouin light scattering spectroscopy of magnetic nanostructures," *IEEE Trans. Magn.*, to be published.

[7] C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, CO, private communication, 2004.

[8] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," *IEEE Transl. J. Magn. Jpn.*, vol. 2, pp. 740-741, August 1987 [*Dig. 9th Annual Conf. Magn. Jpn.*, p. 301, 1982].

[9] M. Young, *The Technical Writer's Handbook.* Mill Valley, CA: University Science, 1989.