一個有效率的不規則相依迴圈平行化技巧
An Effective Parallelization Technique for Non-uniform Loops

平德林
Der-Lin Pean

交通大學資訊工程系
Department of Computer Science and Information Engineering National Chiao Tung University, 1001 TA Hsueh Road Hsinchu, 30050, Taiwan, ROC
dlpean@csie.nctu.edu.tw

陳正
Cheng Chen

交通大學資訊工程系
Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 TA Hsueh Road, Hsinchu, 30050, Taiwan, ROC
cchen@ eicpca5.csie.nctu.edu.tw

摘要

在本論文中，我們提出了一些新的技巧能夠有效的轉換不規則相依迴圈。在這些技巧中，我們利用了迴圈不規則相依的特性來直接而且有效的擷取迴圈的平行度。另一方面，我們的方法也比目前既有的方法有較高的平行度擷取功能。

Abstract

In this paper, we propose a new technique to transform non-uniform dependence loops efficiently, based on their irregularity, by which more parallelism can be explored directly and effectively. Compared with other existing methods, our new approach gives better performance.

關鍵詞：不規則相依性，迴圈的平行化，相依多邊形，平行部份分割、區域平行化分解、不規則迴圈交換、成長樣本偵測。

Keywords: *Non-uniform dependence, loop parallelization, Dependence Convex Hull(DCH), Parallelization Part Splitting(PPS), Partial Parallelization Decomposition(PPD), Irregular Loop Interchange(ILI), Growing Pattern Detection(GPD).*

1. Introduction

According to its cross-iteration dependence relations, a nested loop can be characterized as a uniform or non-uniform dependence loop [6]. Coupled array subscripts in nested loops generate non-uniform data dependence vectors [8]. According to an empirical study on array subscripts and data dependence by Shen et.al. [8], almost 45% of the loops are non-uniform dependence loops in real programs. As a result, parallelization of those non-uniform dependence loops effectively plays an important role in parallelizing compiler design.

However, extracting parallelism of non-uniform dependence loops is very difficult for current parallel compilation techniques. Current techniques can be grouped into two main categories. The first uses the concept of minimum dependence distance to tile non-uniform loops. The other one transforms non-uniform dependence loops into uniform ones. Based on classical convex theory and linear programming techniques [5]-[7], a number of methods have been proposed. One of the method is minimum dependence distance tiling [7]. In this method, the iterations are grouped into tiles which can be executed in parallel. And the tiles can be executed with proper synchronization. A serious disadvantage of this technique is that it tiles iterations on one specific dimension and later on the other dimension in two level nested loops. But the parallelism may be greater if other dimension is tiled first. Minimum dependence vector set [5] also have such disadvantages and may have high complexity. Other approaches based on vector decomposition techniques [6, 9, 10] have also been presented. Those techniques used a set of basic dependence vectors to compose all dependence vectors. They may suffer from extra unnecessary dependences. Hence, all existing approaches have their weakness in parallelizing non-uniform dependence nested loops.

In this paper, we develop a new technique to parallelize non-uniform dependence loops more effective than those of existing non-uniform dependence parallelization methods. Based on the concept of convex spaces[5]-[7], we proposed a new algorithm to extract more parallelisms from the nested loops. Furthermore, our algorithm can enhance not only minimum dependence distance tiling techniques, but it can also enhance other parallelization mechanisms such as minimum dependence vector set methods or uniformization methods. The experiment results show that the performance of our method is better than any previous approach and give very positive insights for further development.

The rest of the paper is organized as follows. In section 2, we introduce the program model considered and review the related work adopted by our

techniques including the dependence analysis process and IDCH computation process. Section 3 discusses our policy for deciding the parallelization mechanisms. The transformed models for effective mechanisms are also given in this section. In section 4, some preliminary performance results collected from our simulation system are presented, and also compared with other similar methods. Finally, some conclusions and future work will be given at the end of the paper.

2. Fundamental background and related work

2.1 Basic concepts[6, 7]

The program model for the loop nest under consideration in this paper is shown in Fig. 1. The dimensions of these arrays are assumed to be equal to the nested loop dimensions. For the simplicity of technique presentation we consider only tightly coupled doubly nested loops. Among the series computation statements, it is assumed that only S_V and S_R can access array A in which $f_1(I,J)$, $f_2(I,J)$, $f_3(I,J)$, and $f_4(I,J)$ are linear subscript expressions of index variable I and J. Dependence exists if S_V and S_R both refer to the same element of array A.

```
      for I = L_I , U_I
           for J = L_J , U_J
S_U:           • • •
S_V:         A(f1(I,J), f2(I,J)) = • • •
                        • • •
S_R:                 • • • = A(f3(I,J), f4(I,J))
            endfor
      endfor
```

Fig. 1. Program Model

In previous research, most loop parallelization techniques are focused on the problem of uniform dependence loops[1]-[4]. However, when the loop is non-uniform dependence loop these methods are either fail or inefficient. As for the non-uniform dependence problem, diophantine equations analysis and IDCH computations are two kernel process for analyzing the data dependence features of non-uniform loops. The general solution of the diophantine equations can be computed by the process proposed by Banerjee [12]. The inequation set given by the program pattern restricts the solution space and forms an area which is called Dependence Convex Hull(DCH) [7].

Lemma 2.1: If the Dependence Convex Hull(DCH) of a nested loop L is empty, then no cross-iteration dependence occurs in L. □

Proof: See [6]□

The DCH can be obtained by the algorithm proposed by Tzen [6]. According to the processes introduced above, several attempts have been made to parallelize non-uniform dependence loops. They focus on two important directions. One direction is dependence uniformization [6, 13, 14] technique, and the other direction is minimum dependence distance [5, 7] technique. We discuss them separately in the following subsections.

2.2 The uniformization technique

Tzen and Ni [6] proposed the first dependence uniformization technique called dependence slope method. They construct two dependence vector sets for two-dimensional loops with non-uniform dependences. For a two-dimensional dependence vector $\overline{d} = [d_1, d_2]^T$, a dependence slope function is defined as d_2/d_1. A pair of upper and lower bounds S_{max} and S_{min} of the slope function can be obtained. Then either $([0,1]^T, [1, \lfloor S_{min} \rfloor]^T)$ or $([0,-1]^T, [1, \lceil S_{max} \rceil]^T)$ is used as Basic Dependence Vectors Set (BDVS). They can compose all non-uniform dependences. However, one of the vector $((0,1),(0,-1))$ has to be in their BDVS, the parallelism of the first dimension will be one and the dependence cone size will remain large.

2.2.1 On uniformization of affine dependence algorithms

Several algorithms have been considered in [13,14] used to uniformize non-uniform dependences based on several objective functions such as schedule length, BDVS cardinalities and dependence cone sizes. Together with six basic ideas and improving techniques, the objective functions are used to guide the selection of the best uniformizations. They enhance the basic uniformization techniques depending on specific dependence patterns. However, they didn't consider dependence directions and their methods are also constrained by the minimum dependence vector of the loop. Our method, on the contrary, can release such constrains by separating the loops and explore more parallelism degrees which will be introduced in the next section.

2.2.2 On effective execution of non-uniform DOACROSS loops

This approach demonstrates a better way to select the uniform dependence vectors and their static strip scheduling schemes enforce dependences with more locality. For a n-dimensional dependence vector $\underline{V} = (\overline{V^1}, \overline{V^2}, \ldots, \overline{V^m})$, where each $V^i = (V^{i,1}, V^{i,2}, \ldots, V^{i,n})$ is a dependence vector in the

n-dimensional iteration space. $\forall i \in \{1,\ldots,m\}$, ub_k^j and lb_k^j are upper and lower bounds of $V_{i,j}/V_{i,k}$ respectively. Then $([1,0]^T,[\lfloor -lb^1{}_2 \rfloor,1]^T)$, $([1,0]^T,[\lfloor -ub^1{}_2 \rfloor,-1]^T)$, $([1,-lb^2{}_1]^T,[0,1]^T)$ and $([1,0]^T,[\lfloor -lb^1{}_2 \rfloor,1]^T, [\lfloor -ub^1{}_2 \rfloor,-1]^T)$ are used as BDVS. They can compose all non-uniform dependences. However, when their BDVS contains and, their algorithms are only successful when $P < M_j$ where M_j is the upper loop bound of parallelized dimension j. When their BDVS contains $\left([1,0]^T,\left[\lfloor lb_2^1 \rfloor,1\right]^T\right)$ and $\left([1,0]^T,\left[\lfloor -ub_2^1 \rfloor,-1\right]^T\right)$, their parallelism is equal to traditional uniformization techniques [6].

In summary, the uniformization based techniques introduce extra dependences and the parallelism is constrained by their BDVS. In order to overcome these difficulties, we introduce another popular parallelization mechanism called minimum dependence distance technique in the next subsection.

2.3 Minimum dependence distance mechanisms[5, 7]

Minimum dependence distance mechanisms use classical convex theory or principles of linear programming to compute the minimum dependence distance and the iteration space to be tiled. Iterations in the same tile can be executed in parellel and the tiles can be executed with proper synchronizations. However, when the dependence vector function do not pass through the IDCH, the non-uniform dependence loop must execute serially and can not explore proper parallelism in the minimum dependence distance tiling technique. In the minimum dependence vector mechanism, they solve integer programming formulations to find the minimum dependence vector set. The minimum dependence vector set is then used to tile the loop for parallel execution.

Our method will enhance the parallelism explored by minimum dependence distance methods in some degree. The above two directions both ignore the irregularity of non-uniform dependence loops. We discover as much parallelism as possible based on the irregularity of non-uniform dependence loops. In the following, we will describe our methods in some detail.

3. Effective non-uniform loop parallelization technique

In this section, we will introduce some mechanisms to improve the parallelism of non-uniform dependence loops. Table 1 shows four popular non-uniform loop models used to evaluate several previous work [5, 6, 13, 17]. We will use them as examples and to evaluate our performance improvement.

At first, we split the loops to get the total parallelization iterations. And then we decompose and transform them to find larger tiling blocks. Finally, we detect some parallelized patterns as large as possible to greatly enhance parallelism of them.

Model 1 [17]	Model 2 [6]	Model 3 [5, 6]	Model 4 [13]

for I=1,N do
for J=1,M do
s1: A(2I, 2J) = ...
s2: ... = A(J+10, I+J+6);
enddo
enddo
	for I=1,N do
	for J=1,M do
	s1: A(I+J, 3I+J+3) = ...
	s2: ... = A(I+J+1, I+2J+4);
	enddo
	enddo
		for I=1,N do
		for J=1,M do
		s1:A(2J+3,I+1) = ...
		s2:.=A(I+J+3, 2I+1);
		enddo
		enddo
			for I=1,N do
			for J=1,M do
			s1: A(3I, 5J) = ...
			s2: ...... = A(I, J);
			enddo
			enddo

Table 1. The standard models.

3.1 Parallelization Part Splitting(PPS)

Because the irregularity of non-uniform dependence loops, total parallelization part of them may occupy most of the iteration space. If we can split such total parallelization parts and execute them in parallel, we can greatly enhance the speedup of non-uniform dependence loops directly. Firstly, we will define the concept of the Non-IDCH region in the following.

Definition 3.1: In the nested loop iteration space Γ, let $i_k(k = 1, 2, 3, \ldots) \in \Gamma$ be the iterations and Γ(IDCH) be the IDCH iteration space. $i'_k \in \Gamma(\text{Non-IDCH})$ if $i'_k \in \Gamma$ and $i'_k \notin \Gamma(\text{IDCH})$. Then the Γ(Non-IDCH) is called the Non-IDCH region of the given nested loop.□

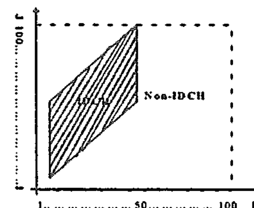

Fig. 3. An example of IDCH and Non-IDCH

Fig. 3 shows the IDCH region and the Non-IDCH region respectively. In this example, most iterations are in the Non-IDCH region. Parallelization of them is very important in this example.

Lemma 3.1: In the nested loop iteration space Γ, $\forall\, i_k, i'_k(k = 1, 2, 3, \ldots) \in \Gamma(\text{Non-IDCH})$, then i_k and i'_k are cross-iteration independent and can be executed in parallel. □

Proof: The proof is done by contradiction.

Let (S1(X,Y), S2(X,Y), S3(X,Y), S4(X,Y)) is the general solution of the corresponding diophantine equations of L. Assume there exists dependence between iteration (i_1, j_1) and iteration (i_2, j_2). There must exist two integers α,β such that (i_1, j_1) = (S1(α,β), S2(α,β)) and (i_2, j_2) = (S3(α,β), S4(α,β)). Since both (i_1, j_1) and (i_2, j_2) are in the iteration space, the following inequalities hold.

$L_I \le S_1(\alpha,\beta) \le U_I$
$L_J \le S_2(\alpha,\beta) \le U_J$
$L_I \le S_3(\alpha,\beta) \le U_I$
$L_J \le S_4(\alpha,\beta) \le U_J$

Thus, (α,β) is in the DCH. Because the IDCH is the set of integer iterations of the DCH, integer point (α,β) is in the IDCH. From definition 3.1, (α,β) is not in the Non-IDCH region. The lemma is proved. □

The theorem 3.1 and 3.2 in the following show how we can split some square part of the Non-IDCH region from the iteration space into appropriate parallelized tiles.

Theorem 3.1: Let L be a doubly nested loop with the form shown in Fig. 1. And let L_I be lower bound of the loop index I and U_J be upper bound of the loop index J. Let L_I(IDCH) be lower bound for the IDCH in loop index I. If $L_I < L_I$(IDCH) then the loop iteration can be split from L_I to (L_I(IDCH)-1) to form a parallel execution tile called Left-tile and the tile size is T_L = (L_I(IDCH) - L_I)*U_J. □

Proof: If $L_I < L_I$(IDCH) then [L_I,(L_I(IDCH)-1)] is in the Non-IDCH region. From Lemma 3.1, iterations in [L_I,(L_I(IDCH)-1)] can be tiled and executed in parallel. Therefore,

T_L = ((L_I(IDCH) -1) - L_I + 1)* U_J = (L_I(IDCH) - L_I)* U_J. □

Theorem 3.2: Let L be a doubly nested loop with the form shown in Fig. 1. And let U_I be upper bound of the loop index I and U_J be upper bound of the loop index J. Let U_I(IDCH) be upper bound for the IDCH in loop index I. If U_I(IDCH) < U_I then the loop iteration can be split from (U_I(IDCH)+1) to U_I to form a parallel execution tile called Right-tile and the tile size is T_U = (U_I-U_I(IDCH))*U_J. □

Proof: If U_I(IDCH) < U_I then [U_I(IDCH)+1, U_I] is in the Non-IDCH region. From Lemma 3.1, iterations in [U_I(IDCH)+1, U_I] can be tiled and executed in parallel. Therefore,

T_U = (U_I-(U_I(IDCH) + 1) +1)* U_J = (U_I-U_I(IDCH))* U_J. □

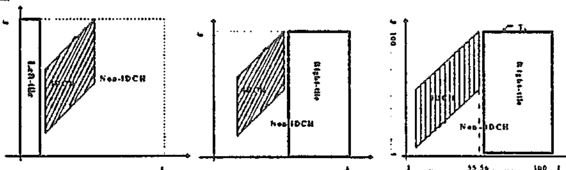

Fig. 4 Tiling of (a) left-tile (b) right-tile (c) example 3.1

Fig. 4(a) shows the Left-tile of Theorem 3.1 and Fig. 4(b) shows the Right-tile of Theorem 3.2. At first, we can execute iterations in the Left-tile in parallel. And then, the iterations which are not tiled are executed using ordinary parallelization mechanisms. Finally, Because the PPS method eliminates all unnecessary dependencies in both Left-tile and Right-tile, the tiled parts after it are not constrained by traditional methods.

Example 3.1: For the program model 1 in table 1 with $U_I = U_J$ =100. Tiling of the Non-IDCH region is shown in Fig. 4(c). We can find that T_L = 0 and T_U = 45*100 = 4500 by using theorem 3.1 and theorem 3.2. According to the PPS mechanism, 45% of the iteration space can be split into the Right-tile in this example and they can be executed in parallel.

```
for I = L_I(IDCH), U_I(IDCH) do
  for J = 1, U_J do
        S_V:     A(f_1(I,J), f_2(I,J)) =...
        S_R:        ... = A(f_3(I,J), f_4(I,J))
  enddo
enddo

DOparallel I = U_I(IDCH)+1, U_I
   DOparallel J = 1, U_J
        S_V:     A(f_1(I,J), f_2(I,J)) =...
        S_R:        ... = A(f_3(I,J), f_4(I,J))
   ENDDOparallel
 ENDDOparallel
```

Fig. 5. Parallel code after using the PPS method

In order to supply standard transformed mechanisms of our methods in the parallelizing compiler, the transformed loop of the program model after using the PPS method is shown in Fig 5. The complexity of the PPS mechanism is restricted by the complexity of the computation of the IDCH region. So the complexity of this algorithm is O(nlogn), where n is the loop bound. However, the parallelization of the IDCH region is still constrained by traditional mechanisms. We will propose effective mechanisms to explore more parallelism from the IDCH in the following sub-section.

3.2 Partial Parallelization Decomposition(PPD)

Parallelism degree explored by existing methods for non-uniform dependence loops is restricted by minimum dependence distance of total iteration space [5-7]. If we can partition the iteration space into different parts and handle them differently, the minimum dependence distance for each part may be larger than original minimum dependence distance. This decompose method exploit more parallelism degree for each partial iteration space. We call it the PPD mechanism, which will be described formally in the following.

Definition 3.2: In the nested loop iteration space Γ and a partial iteration space Γ', $\Gamma' \subseteq \Gamma$. The minimum dependence distance of Γ' is called the Partial Minimum Dependence Distance(PMDD) . □

Definition 3.3: In the nested loop iteration space Γ, the minimum dependence distance of Γ is called the Global Minimum Dependence Distance(GMDD). □

Theorem 3.3: For the nested loop L, $\{p_1, p_2, \ldots, p_n\} \in$ PMDD(L) and $g \in$ GMDD(L) then $p_i \ge g$, $\forall\, i \in 1,\ldots,n$. □

Proof: The proof is done by contradiction. If $\{p_1, p_2,\ldots, p_n\} \in$ PMDD(L) and $g \in$ GMDD(L), we assume that $\exists$ $i \in 1,\ldots,n$ such that $p_i < g$. Because $p_i < g$ and Min{ p_i, g}$\in$GMDD(L), $p_i \in$ GMDD(L). Then $g \notin$ GMDD(L), it contradicts that $g \in$ GMDD(L). Thus $p_i \geq g$, $\forall$ $i \in 1,\ldots,n$. □

In the example, we decompose the iteration space into two sub-iteration space and calculate their partial minimum dependence distance separately. After using our PPD mechanism, the PMDD of the second sub-iteration space is larger than GMDD. The parallelism degree is thus greatly increased.

Fig. 6(a) and (b) show tiles before and after using the PPD method respectively. We can find that tiles of Fig. 6(b) is larger than tiles shown in Fig. 6(a). The transformed loop of the program model is shown in Fig. 7 in which T_i and T_j are tile numbers of the partial region i and j respectively and P_i is the loop bound of first partial block. The transformed loop shown in Fig. 7 have some larger tiles than the tiles before transformation. The complexity of this method is restricted by the complexity of the minimum dependence tiling mechanism, which is O(nlogn), where n is the loop bound. However, the minimum dependence distance may be larger in anothor loop index and the loop interchange may be applied. We will propose a method to detect the validity of interchange for irregular dependence loops and interchange them to extract more parallelism if valid. In the next subsection, we will give detailed descriptions about this technique.

Example 3.2: For the program model 3 shown in table1 with $U_I = U_J = 10$.
$\Gamma = \{ I = 1, 10 \}$, $\Gamma_i = \{ I = 1, 6 \}$, $\Gamma_j = \{ I = 7, 10\}$, $g = 2$, $p_i = 2$, $p_j = 4$. $t = 2$, $t_i = 2$, $t_j = 4$.

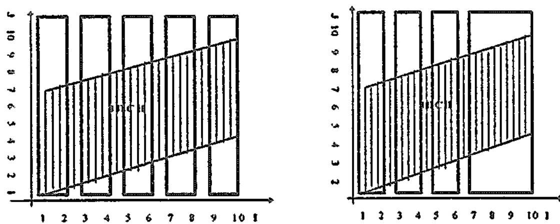

Fig. 6(a) Tile before the PPD method(b)Tile after using the PPD method

```
DOserial K=1, Ti
     DOparallel I = (K-1)* ti +1,min(K* ti, Pi)
        DOparallel J=1, Uj do
Sv:    A(f1(I,J), f2(I,J)) =...
SR:        ... = A(f3(I,J), f4(I,J))
        ENDDOparallel
     ENDDOparallel
ENDDOserial
DOserial K=1, Tj

   DOparallel I = (K-1)* tj +1,min(K* tj , Ui)
      DOparallel J=1, Uj do
Sv:    A(f1(I,J), f2(I,J)) =...
SR:       ... = A(f3(I,J), f4(I,J))
      ENDDOparallel
   ENDDOparallel
ENDDOserial
```

Fig. 7. Parallel code after using the PPD mechanism

3.3 Irregular Loop Interchange(ILI)

We present an effective method to distinguish the validity of non-uniform loop interchange. After using irregular loop interchange, the parallelism of the tiled loop is dependent on the maximum parallelism before and after using the ILI mechanism. In the following, we will briefly survey some previous work and then introduce our ILI mechanism immediately.

Definition 3.4: [15] A dependence $S\ \delta\theta\ S'$ in loop L is c-*interchange preventing* iff S(i/c) δ/c S'(i'/c) does not hold. □

Lemma 3.2: [15] A dependence $S\ \delta\theta\ S'$ in L is c-*interchange preventing* iff $\theta = (=^{c-1}, <,>, *.*)$.□

Lemma 3.3: [15] Loop interchange at level c is valid iff there exists no c-*interchange preventing* dependence. □

We can use the following theorems to distinguish whether interchange is legal or not.

Theorem 3.4: Assume that the general solutions of dependence vectors which are interchanged are $V_c(X, Y)$ and $V_{c+1}(X, Y)$ respectively. A dependence $S\ \delta\theta\ S'$ exists in L where $\theta = (=^{c-1}, V_c(X, Y), V_{c+1}(X, Y), *\ldots*)$. Loop interchange at level c is valid iff $V_c(X, Y)* V_{c+1}(X, Y) \geq 0$.

Proof: Let the assumption of the theorem hold. We apply Lemma 3.3. Loop interchange at level c is valid then there exists no c-*interchange preventing* dependence. From Lemma 3.2, a dependence $S\ \delta\theta\ S'$ in L then $\theta \neq (=^{c-1}, <,>, *\ldots*)$. Clearly, $V_c(X, Y)* V_{c+1}(X, Y) < 0$ can not be satisfied. We have $V_c(X, Y)* V_{c+1}(X, Y) \geq 0$.

Conversely, $V_c(X, Y)* V_{c+1}(X, Y) \geq 0$ implies that a dependence $S\ \delta\theta\ S'$ in L then $\theta \neq (=^{c-1}, <,>, *\ldots*)$, which concludes the proof of the theorem. □

Theorem 3.5: Assume that loop interchange at level c is valid. The loop bounds for the normalized loop index c and c+1 are U_c and U_{c+1} respectively. And the minimum dependence distance of loop index c and c +1 are $Min(d_c)$ and $Min(d_{c+1})$ respectively. The maximum parallelism with minimum dependence tiling for loop index c and c +1 is $Max(Min(d_c)* U_{c+1}, Min(d_{c+1})* U_c)$.

Proof: Let the assumption of the theorem hold. There is no dependence between any two iteration inside the tile with tile length $Min(d_c)$ and $Min(d_{c+1})$ respectively. The iterations of the tile after using tiling are $Min(d_c)* U_{c+1}$ and $Min(d_{c+1})* U_c$. The maximum parallelism of loop index c and c + 1 are the maximum tile size of loop index c and c + 1. Then the maximum parallelism are $Max(Min(d_c)* U_{c+1}, Min(d_{c+1})* U_c)$. □

Theorem 3.4 detects whether some specific kinds of irregular loop can be interchanged or not. If it is valid after detection, we can interchange the irregular loops immediately. Theorem 3.5 tells how to calculate the minimum dependence distance before and after using the ILI method and choose the larger minimum dependence distance to be the tile length. In order to illustrate our ILI method clearly, let's consider the following example.

Example 3.3: For the program model 4 with $U_I = U_J = 10$, Dependence Vector = (2X, 4Y), $V_1 * V_2 = 2X*4Y > 0$, $Min(d_1) = 2$, $Min(d_2) = 4$

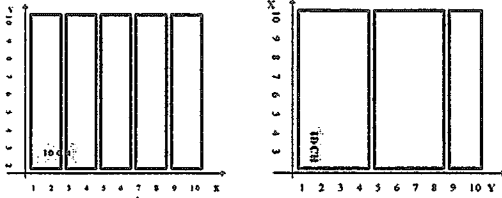

Fig. 8(a). Tile before the ILI mechanism (b). Tile after using the ILI mechanism

```
DOserial K=1, Ti
    DOparallel J = (K-1)* tj +1,min(K* tj +1, Uj)
        DOparallel I=1, Ui do
          Sv:  A(f1(I,J), f2(I,J)) =...
            SR:         ... = A(f3(I,J), f4(I,J))
        ENDDOparallel
    ENDDOparallel
ENDDOserial
```

Fig. 9. Parallel code for the ILI mechanism

In Fig. 8(a), the maximum parallelism before the ILI mechanism is 20. It is 40 after applying the ILI mechanism as shown in Fig. 8(b). The complexity of this method is dominated by the complexity of the minimum dependence distance tiling mechanism. So it is O(nlogn) where n is the loop bound. The transformed loop of the program model is shown in Fig 9. T_i and T_j are tile numbers of the loop index i and j respectively.

On the other hand, there exists some regular dependence vector patterns in some loop indices. In the next subsection, we will introduce a method to compute the minimum dependence distance of those regular dependence patterns efficiently.

3.4 Growing Pattern Detection(GPD)

If the dependence vector function is increasing or decreasing, we can tile the loop according to the dependence vector function. The minimum dependence distances of the loop index can be computed according to functions proposed in the following to extract more parallelism.

Definition 3.5 Growing Pattern: Let dependence vector function V_I be real on (L_I, U_I). Then the pattern of V_I is said to be growing pattern on (L_I, U_I) if $L_I < x < y < U_I$, $\forall x, y \in \mathbb{Z}$, implies $|V_I(x)| \le |V_I(y)|$.□

Lemma 3.4: Let I and J be index variables. The pattern of the dependence vector is the growing pattern if the dependence vector function V_I and V_J contain only the linear function of loop index variable I and J respectively.□

Proof: If the dependence vector contains only one variable I, it must be $a*I + c$, where $a,c \in \mathbb{R}$. We can differential it, then $d(a*I + c)/dI = a$. Clearly, $|a*x + c| \le |a*y + c|$, $\forall x, y \in \mathbb{Z}$ and $x < y$. Thus, the pattern of the dependence vector function V_I is the growing pattern. The dependence vector function V_J can also be proven as growing pattern by the similar way.□

Lemma 3.5: Assume the pattern of dependence vector function V_I for loop index I is the growing pattern. The maximum tile length T of the loop index variable $i \in I$ will be $|V_I(i)|$.

Proof: Because the pattern of dependence vector function V_I for loop index I is the growing pattern,. $\forall x, y \in \mathbb{Z}$, implies $|V_I(x)| \le |V_I(y)|$. The maximum tile length T of the loop index variable $i \in I$ must be $|V_I(i)|$, because $|V_I(i)| \le |V_I(x)|$, $\forall x \in (i + 1, i + |V_I(i)| - 1)$.□

Theorem 3.6: Assume the pattern of dependence vector function V_I for loop index I is the growing pattern and the dependence vector function $V_I = a*I + c$, where $a,c \in \mathbb{R}$, I is the variable of the corresponding loop index. The tile size T_n for tile n is $|a*(S_{n-1}+1)+c|$, where S_n is equal to $|\sum_{i=0}^{n-1}(a+1)^i(a+c)|$.

Proof: The theorem can be proven by induction on k. We begin our induction at 1.

Basis k = 1. Then T_1 is equal to the absolute value of dependence vector $V_I(1)$. $T_1 = |V_I(1)| = |a + c|$.

Induction k > 1. By the induction hypothesis, $T_k = |a*(S_{k-1}+1)+c| = |a*|\sum_{i=0}^{k-2}(a+1)^i(a+c) +1| +c|$. We must show that $T_{k+1} = |a*(S_k+1)+c| = |a*|\sum_{i=0}^{k-1}(a+1)^i(a+c) +1| +c|$. Then as

$T_{k+1} = V_I(Sk+1) = |a*(S_k+1) + c|$. We have

$$T_{k+1} = |a*(S_{k-1}+T_k+1)+c| = |a*(S_{k-1}+a(S_{k-1}+1)+c+1)+c|$$
$$=|a*((a+1)S_{k-1}+a+c+1)+c|$$
$$=|a*(a+1)(|\sum_{i=0}^{k-2}(a+1)^i(a+c)|+a+c+1)+c|$$
$$=|a*[|\sum_{i=0}^{k-2}(a+1)^{i+1}(a+c)|+(a+c)+1]+c|$$
$$=|a*[|\sum_{i=0}^{k-1}(a+1)^i(a+c)|+1]+c| = |a(S_k+1)+c|$$
$$= V_I(Sk+1)$$

Then the theorem is proven. □

Example 3.4: For the program model 4 with $U_I = U_J = 10$, Dependence Vector = (2X, 4Y).

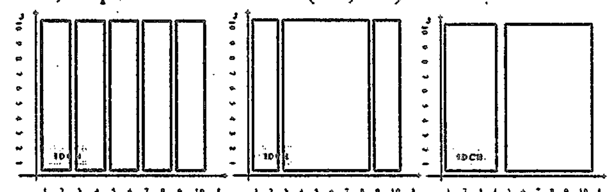

Fig.10(a)Original Tiling.(b)the GPD method.(c)the ILI and GPD.

In Fig. 10(a), minimum dependence distances are static and equal to 2. In Fig. 10(b) and (c), they are increasing and greater than original tiles. The average tile size after applying the ILI and GPD methods is greater than the GPD method only. The transformed loop of the GPD method is shown in Fig. 11(a). The tile size of the GPD method can be calculated by using theorem 3.6. In Fig. 11(b), we also show the transformed loop of the combination of both the GPD and the ILI methods. T_i and T_j are tile numbers of the dimension i and j respectively. S_k is the sum of the tile sizes from T_1 to T_k. The complexity of this algorithm is dominated by the computation of T_k and is O(n) where n is the loop bound.

```
DOserial K=1, T_i
  DOparallel I=S_{k-1}+1, min(S_k, U_i)
      DOparallel J=1, U_j do
S_V:    A(f_1(I,J), f_2(I,J)) =...
S_R:       ... = A(f_3(I,J), f_4(I,J))
      ENDDOparallel
  ENDDOparallel
ENDDOserial
```

Fig. 11. (a) Parallel code for the GPD method.

```
DOserial K=1, T_j
  DOparallel J= S_{k-1}+1, min(S_k, U_j)
    DOparallel I=1, U_i do
    S_V:       A(f_1(I,J), f_2(I,J)) =...
S_R:        ... = A(f_3(I,J), f_4(I,J))
    ENDDOparallel
  ENDDOparallel
ENDDOserial
```

Fig. 11.(b) Parallel code for the ILI and GPD methods.

```
Algorithm Effective Parallelization mechanisms:
input: The nested loop L(I, J) with irregular dependence
output: Transformed loop with combination of our parallelization mechanisms
begin
  Let V_I and V_J be the dependence vector functions of the nested loop L(I, J);
  Calculate the diophantine equation f(L) of the nested loop L;
  Calculate V_I and V_J according to f(L);
  Build the IDCH(L) according to f(L);
  ILI Detection(ILID(L)) variable = False;
  If ILID(L) = true then
    L' = ILI(L); /* L' is transformed after applying ILI mechanism */
    L'' = GPD(L');
    Parallelism degree is Speedup-ILI, GPD;
    /* Speedup-ILI, GPD is speedup after ILI and GPD */
    L'' = PPS(L');
    L''' = PPD(L'');
    Parallelism degree is Speedup-ILI, GPD, PPS, PPD;
  else
    L' = GPD(L);
    Parallelism degree is Speedup-GPD;
    L' = PPS(L);
    L'' = PPD(L');
    Parallelism degree is Speedup-GPD, PPS, PPD;
  endif
        Speedup-all = Max((Speedup-ILI, GPD), (Speedup-ILI, GPD,
        PPS, PPD), (Speedup-GPD), (Speedup-GPD, PPS, PPD));
end
```

Fig. 12 The algorithm of Effective Parallelization mechanisms

In summary, different mechanisms can not only be applied individually, but also can be applied with different combinations. The algorithm in Fig. 12 depicts the whole procedures of our effective parallelization mechanisms. The symbols used in Fig.12 is presented in table 2.

4. Preliminary performance evaluations

Preliminary performance evaluations of proposed mechanisms is presented in this section. We assume that the processor number is unlimited and the speedup is calculated by the parallelism degree. We compare our performance evaluations with two traditional non-uniform parallelization methods, named minimum dependence distance tiling mechanism [7] and dependence uniformization mechanism [6]. Fig. 13 shows the performance results of our techniques by calculating maximum parallelism degree before and after transformations proposed.

In each experiment, different loop bounds are applied to a doubly nested loops. Two basic parallelization techniques are implemented, because they explore most parallelism of traditional methods. One is the uniformization method and the other is the minimum dependence distance tiling method. Fig. 13 not only shows speedup between the uniformization method and the minimum dependence distance tiling method, but also shows the speedup of our effective mechanisms. The minimum dependence distance tiling methods usually perform better than the uniformization method because the later introduces extra dependences. Fig. 13(d) illustrates it.

SYMBOL	U	T	PPS	PPD	ILI	GPD
Meaning	Uniformization	Minimum Dependence Distance Tiling	Parallelization on Part Splitting	Partial Parallelization Decomposition	Irregular Loop Interchange	Growing Pattern Detection

Table 2. The meaning of the symbols used in Fig. 12 and Fig. 13.

Fig. 13(a) and Fig. 13(b) show the great speedup of our PPS method, because they have most part can be split for total parallelization. Fig. 13(b) and Fig. 13(c) show the uniformization method performs better than the minimum dependence distance tiling method because the uniformization method provides another techniques when two dependence vectors both go through the IDCH. Fig. 13(b) and (c) shows performance of the PPD method. Fig. 13(d) shows that the ILI method can

be adapted because interchange is valid in this program model. Fig. 13(c) and Fig. 13(d) both have growing pattern in their dependence vector patterns. In Fig. 8, when the loop bounds is increased from 5 to 50, the speedup of our PPS technique is increased from 1 to 1.8 times better than the uniformization technique [6] and from 1 to 2 times better than the minimum dependence distance tiling technique [7]. Our PPD technique is 1.42 to 2 times better than the minimum dependence distance tiling technique. Our ILI technique is 1.5 to 2 times faster than the minimum dependence distance tiling technique. The speedup of our GPD technique is increased from 3.25 to 19.8 times better than the uniformization technique and from 1.25 to 5 times better than the minimum dependence distance tiling technique. Combination of our ILI technique and GPD technique is 1.5 to 8.33 times better than the minimum dependence distance tiling technique. In summary, our effective techniques is 3.25 to 19.8 times faster than that of the uniformization technique and 1.5 to 8.33 times better than that of the minimum dependence distance tiling technique. From above figures, the different mechanisms can be combined to extract much more parallelisms. The efficiency of different mechanisms depend on their program patterns. When our effective mechanisms are implemented at those popular models, we can explore much more parallelism degree than those of traditional methods

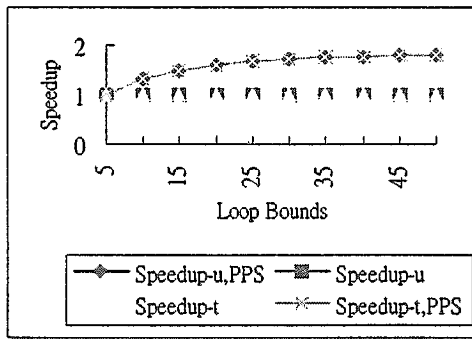

Fig. 13(a) Speedup for Model 1

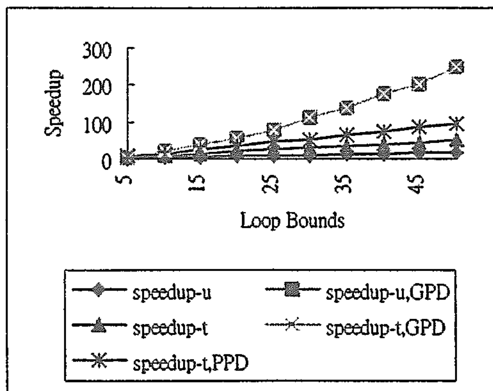

Fig. 13(c) Speedup for Model 3

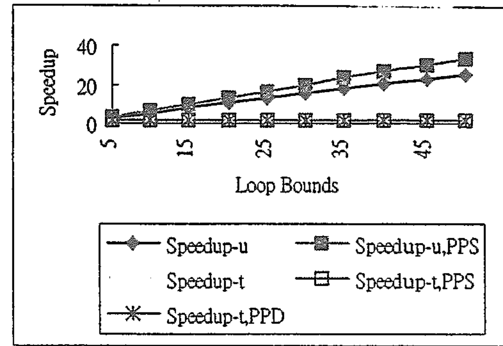

Fig. 13(b) Speedup for Model 2

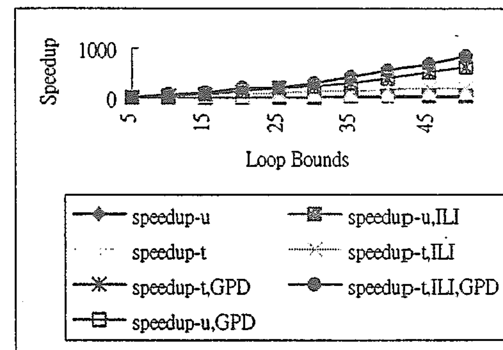

Fig. 13(d) Speedup for Model 4

5. Concluding remarks

In this paper, we present an effective way to enhance the parallelism of non-uniform dependence loops. Several mechanisms, including PPS, PPD, ILI, and GPD are proposed to enhance different program models and they also can be combined to enhance the same program model. We also compare performance of traditional parallelization and our parallelization techniques. When the loop bounds is increased from 5 to 50, our effective techniques is 3.25 to 19.8 times faster than the uniformization technique and 1.5 to 8.33 times better than the minimum dependence distance tiling technique. The speedup of our enhanced mechanisms shows that our mechanisms can potentially explore parallelism of current popular models. We have built most parts of our mechanisms in our parallelizing compiler environments based on SUIF [16]. In the future, further enhancement and performance evaluations will be going on.

References

[1] Alain Darte and Yves Robert, Member, IEEE, "Constructive Methods for Scheduling Uniform Loop Nests ", IEEE Transactions on Parallel and Distributed System VOL. 5, NO. 8, pp. 814-822, Aug. 1994.

[2] Leslie Lamport "The Parallel Execution of DO Loops " Communications of the ACM ,Vol. 17 No. 2, pp. 83-93, Feb. 1974.

[3] Erik H. D'Hollander, "Partitioning and Labeling of Loops by Unimodular Transformations" IEEE Transactions on Parallel and Distributed System Vol. 3, No.4, pp. 465-476, July. 1992.

[4] Marsha J. Berger. "A Partitioning Strategy for Nonuniform Problems on Multiprocessors", IEEE Transactions on Computers Vol. C-36, NO. 5, pp. 570-580, May. 1987.

[5] Shaw-Yen Tseng, Chung-Ta King and Chuan-Yi Tang "Minimum Dependence Vector Set: A New Compiler Technique for Enhancing Loop Parallelism" In Proc. of 1992 ICPADS, pp. 340-346, Dec. 1992.

[6] Ten H. Tzen and Lionel M.Ni "Dependence Uniformization: A Loop Parallelization Technique", IEEE Transactions on Parallel and Distributed System Vol. 4, pp. 547-558, May. 1993.

[7] Swamy Punyamurtula and Vipin Chaudhary "Minimum Dependence Distance Tiling of Nested Loops with Non-uniform Dependences", Proceeding of 6th IEEE Sym. on Parallel and Distributed, pp. 74-81, May. 1994.

[8] Zhiyu Shen et al. "An Empirical Study on Array Subscripts and Data Dependencies " Proc. of the 1989 ICPP IEEE, pp. 145-152, 1989.

[9] F. Irigoin and R. Triolet, "Supernode partitioning," in Conference Record of the 15th ACM Symposium on Principles of Programming languages, pp. 319-329, 1988.

[10] J.Ramanujam and P.Sadayappan "Tiling of multidimentional iteration spaces for multi-computers" Journal of Parallel and Distributed Computing , Vol.16, pp. 108-120, Oct. 1992.

[11] M. S. Bazaraa, J. J. Jarvis, and H.D. Sherali, *Linear Programming and Network Flows*, John Wiley & sons,1990.

[12] U.Banerjee, *Dependence Analysis for Supercomputing*, Kluwer Academic, 1988.

[13] Weijia Shang, Edin Hodzic and Zhigang Chen "On Uniformization of Affine Dependence Algorithms" , IEEE Trans. On Computers, Vol. 45, No. 7 , pp. 827-840, July 1996.

[14] Ding-Kai Chen et al."On Effective Execution of Nonuniform DOACROSS Loops", IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 5, May. 1996.

[15] Hans Zima, *Supercompilers for Parallel and Vector Computers*, ACM press Frontier Series, Addison Wesley 1990.

[16] Robert Wilson, Monica Lam, and John Hennessy et al., An Overview of the SUIF Compiler System, Computer Systems Lab Stanford University, 1996.

[17] Shaw-Yen Tseng, Chung-Ta King and Chuan-Yi Tang "Profiling Dependence Vectors for Loop Parallelization", Proceedings of IPPS, pp. 23-27, 1996.