一個有效不規則相依迴圈計算與資料分解方法之研製

平德林，李佳霖及陳正

國立交通大學資訊工程系

新竹市大學路 1001 號，30050

Tel:(886)35712121 EXT:54734, Fax:(886)35724176

E-mail: cchen{dlpean}@csie.nctu.edu.tw

摘 要

由於遠端記憶體存取 (Remote memory access) 時間遠大於區域記憶體存取(Local memory access)。因此，我們利用計算分解 (communication decomposition) 與資料分解 (data decomposition) 之架構計算出迴圈中計算 (iteration)與陣列資料之相關性。並採用廣域資料分析，對整個程序中之迴圈進行陣列資料分析，促使陣列資料分解型態一致，以減少資料重組溝通。同時利用迴圈交換 (loop interchange) 方法，促使迴圈中之計算對陣列資料之存取順序能符合資料之區域性 (data locality)。最後，經由資料型態之分析決定出最後之區塊或循環分配方式，將具有相關性之計算與資料分配到同一個處理器和其區域記憶體中。初步評估結果顯示，當陣列之下標函數之變數較為複雜時，本方法能經由計算與資料分解之分配方式，降低遠端資料存取順序。而當迴圈之計算順序會破壞陣列資料之區域性時，本方法會利用迴圈交換方式，使其存取順序滿足資料之區域性，進而降低遠端存取次數。因此，將此方法搭配不規則相依迴圈靜態排程方式，確實可減少迴圈中遠端存取之次數。

關鍵字: 計算分解，資料分解，不規則相依迴圈，靜態排程，平行編譯器。

第一章 緒 論

隨著多處理機系統的發展，具Non-Uniform Memory Access time (NUMA)特性的分散式共享記憶體多處理機系統 (Distributed-Shared Memory Multi-processor System)愈來愈受重視，它兼顧了分散與共享記憶體的特性 [9]。在NUMA系統，每個處理機都有自己的記憶體模組 (Local Memory)，其他的則為遠端記憶體模組 (Remote Memory)。遠端的 (Remote)資料存取時間會比區域性的 (Local)資料存取時間長很多；所以如果資料放置不恰當，系統會花很多時間在傳遞資料，嚴重影響系統效能。因此，如何依據程式的計算 (Computation)以及資料依存關係，將資料 (data)適當地分配 (Decomposition)到各個處理機上，以達到系統的最佳效能，是平行編譯器的一個重要課題 [1]。

目前，這方面的研究大多是探討在規則資料相依迴圈 (Uniform Data Dependence Loop)[3]中的資料分配 [1, 2, 6, 12]。但是在不規則資料相依迴圈 (Non-uniform Data Dependence Loop) [13]中對於資料分配的研究卻很少。原因在於規則資料相依迴圈在做資料分配時不必考慮資料和位置的關係。反觀在不規則資料相依迴圈中資料相依關係會因迴圈所在位置而有差異。因此，資料分配上就變的非常複雜。以往遇到較為複雜的資料相依向量時，就必須使用動態分配的方法，但分配但分配動作會增加程式的執行時間 [1]。若無法有效分配資料，反而會降低整體的效能。因此，我們採用靜態分配來分配迴圈中陣列之資料。

我們針對在不規則資料相依迴圈中資料相依關係、迴圈所在位置及資料使用形態 (pattern)加以觀察，找出不規則資料相依迴圈中資料分配的方法。再依據迴圈位置排程來決定位置分配函數 (iteration decomposition function)[2]，並利用整體平行度與區域性最佳化 (global optimizations for parallelism and locality)[2]方法，求出相對的資料分配函數(data decomposition function)。接著將程序中每個迴圈所對應出的資料分配函數，進行廣域(global)之整合分析，並配合區塊 (block)或是環狀 (cyclic)分配 [4]方式，將陣列資料依據資料分配函數配置 (allocation)於分散式共享記憶體系統中每個處理機的記憶體模組中。同時，我們也將本論文所提之方法實作在由Stanford Compiler Group所發展，名為SUIF [14]的一套平行編譯環境，並將編譯後的結果在CONVEX SPP 1000分散式共享記憶體多處理機系統進行模擬評估。

我們利用不規則相依迴圈靜態排程方法中之一種索引同步方法 (Index Synchronization Method) [13]為我們評估之靜態排程方式。並利用其方法所採用之二個程式模型 (Program model)和實際應用程式 Fishpack [16]、Eispack [17]和Linpack [18]中之四個不規則相依迴圈為評估對象。根據評估結果，本方法的確能減少遠端記憶體存取次數。

第二章將對整個研究背景做介紹。第三章則是我們所提出之不規則資料相依迴圈中資料的分配方法。第四章將介紹所提方法在平行編譯環境中實作過程與分析驗證。最後在第五章將會對整個研究做結論並列出未來可能的研究方向。

第二章 相關研究背景概述

我們先介紹目前巢狀迴圈中資料分配的相關技術，定義規則與不規則資料相依迴圈，接著再定義我們研究的問題與問題的假設。

2.1 相關研究方法回顧[1, 5, 10, 12]

在平行多處理機環境中，將一個程式的計算工作切割、分配到各處理機上，稱為這個程式的計算分配；同樣地，將一個程式所用到的資料分配到各處理機的記憶體上，則稱為這個程式的資料分配。為了能夠善用多處理機平行運作的特性，我們必須將程式及所使用到的資料作適當的切割、分配，使之能夠發揮平行處理的最大效益，由於程式中，迴圈所佔的相對執行時間比例相當大，而且由於迴圈重覆執行類似工作的特性，所以迴圈成為平行化最好的對象[3]。因此我們所探討的分配技術是以迴圈中的陣列為對象。

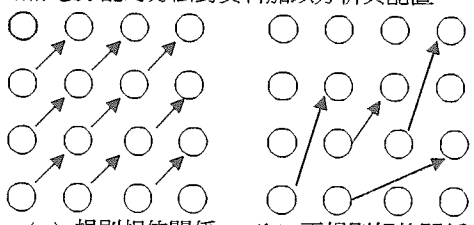
在規則資料相依迴圈的資料分配方法方面，初期的研究，只針對單一迴圈爲處理對象，稱爲區域分析法(Local Analysis)。這方面的研究有許多，其中如J.Ramanujam [12]所提出的一種與機器架構無關的分析方法，利用矩陣運算，找出無溝通超平面(Communication-Free Hyperplane)的資料分配；M.E.Wolf[15]則針對完全巢狀迴圈(Perfectly-Nested Loop)，將迴圈作轉換(Loop Transformation)以提高較大顆粒(Coarse-grain)的平行度；另外，亦有利用方塊法(Tiling)來開發資料的區域性[15]。其他的相關研究還有[7, 8]等等。但是實際上的應用程式，不只有一個迴圈，區域分析法無法滿足需要。必須同時考慮不同迴圈間的關係，稱爲廣域分析法(Global Analysis)[5]。

C.H.Huang[5]將超平面切割擴展至以數個迴圈同時爲處理對象。他們提出了找出無溝通超平面切割(Communication-Free Hyperplane Partition)方法，在無法找出單一超平面切割(Single-Hyperplane Partition)時，有一些情況他們能找出多個無溝通超平面切割。但是，在無法找出無溝通切割時，沒有提出解決辦法。

J.M.Anderson and M.S.Lam [1]先利用 [15]的區域分析找出每個迴圈的可完全平行迴圈，接著對整個程式作處理。它是以線性代數的向量空間(Vector Space)的作法來找切割方式，並以線性轉換矩陣表示迴圈與陣列切割的結果。如果每個陣列在不同的迴圈中有相同的分配方式，則稱爲靜態分配 [1]；若否，陣列在不同的迴圈中有不同的分配方式，則在進入另一迴圈時，需要作陣列資料搬移的工作，稱爲動態分配 [1]。如果能夠找到靜態分配，則資料傳輸時間最小。

規則資料相依迴圈的資料分配方法中，有一個共通的缺點，就是當無法找到無溝通切割或是靜態分配的方法時，就只好以動態分配的方法來找尋最佳解，然而動態分配卻是NP-Hard，[1]。其無法找到無溝通切割或是靜態分配的原因是資料之間的相依關係。因爲上述的各種方法，只能在資料相依向量規則的情況之下運行；一旦遇到較爲複雜的資料相依向量時，就會無法找出無溝通情況，而必須使用動態分配的方法。若無法有效分配資料，反而會降低系統整體效能。

又因不規則資料相依迴圈中，資料相依向量的複雜度比較高。因此關於不規則資料相依迴圈中資料分配的方法也就非常的少。所以我們擴展 J.M.Anderson 和 M.S.Lam[1]計算與資料分解數學方法，找出不規則資料相依迴圈中計算分解函數所對應的資料分解函數，並利用靜態分配的方法對資料加以分析與配置。



(a) 規則相依關係　(b) 不規則相依關係

圖1　迴圈空間(Iteration spaces)與相依關係向量

2.2 問題定義與假設

圖1(a)與1(b)分別代表兩個不同的雙層巢狀迴圈，其內之有向向量即表示出其各自的資料相依關係。我們的資料分配方法即爲針對不規則相依迴圈問題所提出的技術。圖1爲規則資料相依關係(圖1.a)與不規則資料相依關係迴圈(圖1.b)的例子，圖中的節點表示各迴圈主體而向量則代表相依向量。

我們所討論的問題將以兩層的巢狀迴圈(2-dimentional nested loop)爲對象。圖 2 即爲本論文所討論的兩層迴圈之標準型式；其中，Ub_1 、Ub_2 爲常數、S_1與 S_2爲迴圈主體內參考同一陣列資料的兩個敘述而 g_1, g_2, g_3, g_4皆爲索引變數 I,J 的線性函數(linear function)。

```
Do I = 1, Ub1
  Do J = 1, Ub2
   S1:  A[g1(I,J)][g2(I,J)] = .............
   S2:   ............. = A[g3(I,J)][g4(I,J)];
  enddo
enddo
```

圖 2　對象迴圈型式

第三章　不規則相依迴圈計算與資料分解

3.1 計算與資料分解基本概念

本節將介紹計算與資料分解的數學架構。我們將整個分解的過程分成二個部份。第一部份是將計算和資料對應到虛擬處理器空間(virtual processor space)。第二部份則是將此虛擬處理器空間對應到實際機器的處理器中。

3.1.1 虛擬處理器空間轉換 [2]

對一個 *l* 層的迴圈，定義其迴圈空間爲 I，且迴圈中的每個迴圈空間均以指標向量 $i=(i_1, i_2, \ldots, i_l)$表示。對一個具有 m維空間的陣列，定義其陣列空間爲 A，且陣列中的每筆資料均以陣列指標向量 $a=(a_1, a_2, \ldots, a_m)$來表示。同理，對於具有 n 維的處理器空間，其處理器空間定義爲 P。另外，將陣列的下標函數表示成仿射(affine)陣列存取函數 $f: I \rightarrow A$, $f(i) = Fi+\zeta$，其中 F是一個線性的轉換式，ζ 則是一個常數向量。同時也將計算與資料分解對應到虛擬處理器空間的對應函數稱爲仿射計算分解與仿射資料分解。而其定義分述如下：

定義 1:　*若將 m 維空間陣列的指標向量表示成 $a=(a_1, a_2, \ldots, a_m)$。則仿射資料分解是指將陣列分解對應到 n 維的處理器空間，而其仿射函　$d: A \rightarrow P$ 爲 $d(a) = Da + \delta$，其中 D 是 n*m 的線性轉換矩陣，δ則是一個常數向量。*

定義 2:　*若將 l 層迴圈的指標向量表示成 $i=(i_1, i_2, \ldots, i_l)$。則仿射計算分解是指將迴圈中的 iteration 分解對應到 n 維的處理器空間，而其仿射函　$c: I \rightarrow P$ 爲 $c(a) = Ci + \gamma$，其中 C 是 n*l 的線性轉換矩陣，γ則是一個常數向量。*

同時也稱仿射分解中線性轉換矩陣部份爲線性分解(linear decomposition)，而其常數向量部份稱爲位移分解(offset decomposition)。也就是跟據前述之定義，D和 C 分別爲線性資料與計算分解；而δ和γ則分別爲位移資料和計算分解。

圖 3(a)表示二維陣列仿射資料分解到一維虛擬處理器。圖 3(b)則是表示二層迴圈仿射計算分解到一維虛擬處理器。陰影的部份表示資料所在位置。迴圈中計算與資料的關係來自於陣列中的存取函數 (access function)。因此若能將仿射陣列存取函數和仿射計算分解函數、仿射資料分解函數作數學運算，即可找出計算與資料分解之關係式，其過程詳述於後。

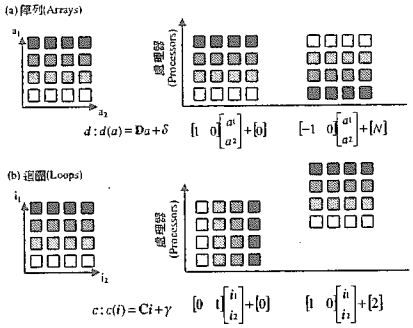


圖 3 仿射分解對應虛擬處理器空間例子:(a)仿射資料分解和(b)仿射計算分解。陰影則表示資料存放位置

對巢狀迴圈 j，將其仿射計算分解表示爲 $c_j(i)=C_j(i)+\gamma_j$。並將迴圈中 x 陣列仿射資料分解表示爲 $d_x(a)=D_x(a)+\delta_x$。而迴圈 j 中 x 陣列的第 k 個陣列存取函數則表示爲 $f_{x,j,k}$。如果迴圈 j 中 x 陣列的仿射計算分解和仿射資料分解符合下列的數學式子，則表示不會有溝通(communication)情況。

$$D_x(f_{x,j,k}(i)) + \delta_x = C_j(i)+\gamma_j \qquad (1)$$

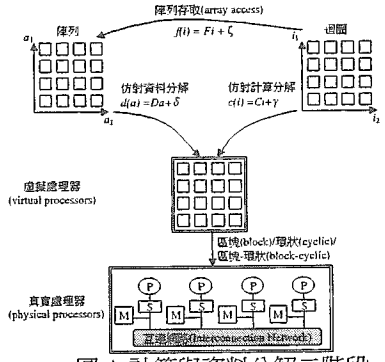


圖 4 計算與資料分解二階段

如果溝通的發生是由於線性分解所引起時，則溝通花費將會非常的多，因爲必須對整個陣列做資料重組。相對地，如果溝通是由位移分解所引起時，就如同最近相鄰溝通，其花費並不多。因此，將式子 1 中位移分解的部份忽略，同時 $f_{x,j,k}(i)=F_{x,j,k}(i)+\zeta_{x,j,k}$，則可得出如式子 2 所示之數學式。

$$D_x F_{x,j,k}(i) = C_j(i) \qquad (2)$$

3.1.2 實際處理器轉換 [2]

找出仿射計算與資料分解所對應的虛擬處理器之後，接著便利用區塊(block)、環狀(cyclic)或是區塊-環狀(block-cyclic)等三種分配函數，將虛擬處理器分配到真實處理器中。所謂區塊分配函數是指將$\lceil P_v/P_p \rceil$個連續的虛擬處理器分配到每一個實際處理器中，其中 P_v 是指虛擬處理器的個數，而 P_p 則是指實際機器的處理器個數。環狀分配函數則是指利用循環環繞(round-robin)的方式將每個虛擬處理分配給每一個實際的處理器。而區塊-環狀(b)分配方式則是融合了區塊與環狀分配的方法，將 b 個連續的虛擬處理器以循環環繞方式分配給處理器。因此，完整的分解包含了二個階段，第一是虛擬處理器的轉換仿射函數，第二則是實際處理器的分配函數。如圖 4 所示。

3.2 不規則迴圈計算與資料分解

現在介紹不規則相依迴圈中計算與資料分配方法。我們也採用計算與資料分解二階段的概念，至於原本的函數[2]都無法應用在不規則相依迴圈中。

由於我們的對象迴圈爲二層不規則相依迴圈且陣列型式爲耦合性下標陣列，所以可以將迴圈空間和陣列空間表示爲二維空間。而計算與資料分解數學式則表示迴圈空間和陣列空間的一次線性方程式。在不規則相依迴圈中就無法依一定規則來分配。所以我們的方法輔以靜態的排程(static schedule)技術[13]找出每個規則或可平行化的區域，再對這些區域作虛擬分配到實際處理器的轉換。

3.2.1 計算分解函數

我們可以將仿射計算分解函數表示成 $c_k(i)=C_k(i)+\gamma_k$，其中 k 表示程序中第 k 個巢狀迴圈。C_k 爲 1*2 矩陣，稱之爲計算分解矩陣。而 γ_k 則代表位移常數。藉由靜態排程的單位切割方式，可決定出 C_k 的值，同時給予每個切割單位一個虛擬處理器識別碼(P_{vid})。圖 5(a)(b)(c)分別表示以第一層迴圈(I)、第二層迴圈(J)和直線斜率爲 1 的單位切割排程方式和其對應的仿射計算分解函數，以及每個切割單位所分配的虛擬處理器(P_{vid})。陰影表示虛擬處理器位置。

[範例 1]

```
Do I = 1, 10
   Do J = 1, 10
      A( 2J, 2I ) = A( J, I )
   Enddo
Enddo
```

以範例 1 爲例並利用索引同步方法(Index Synchronization Method)[13]的排程方式。得知可平行化區域個數爲 1，而此區域的可平行化單位個數爲 10，且此區域中相依向量爲{(0,1), (1,0)}，其相依關係如圖 6 所示。其中灰色區塊表示分配的基本單位，也就是此區塊中的 iterations 都會分配到同一個處理器中。依同步索引排程方式，可以定出此迴圈計算分解矩陣爲[1 0]，而位移常數爲-1。也就是將第一層索引值爲 i 的 iterations 都放在識別碼爲(i-1)的虛擬處理器中。

圖 5 迴圈排程相對應之計算分解函數

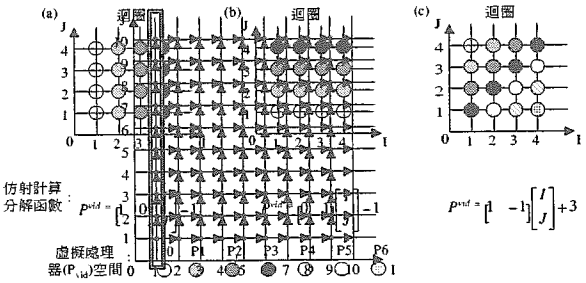


圖 6 範例 1 經同步索引方法排程後的相依向量

3.2.3 陣列存取函數

找出迴圈中計算分解函數後，接著便將迴圈中陣列的存取型態(access pattern)表示成陣列存取函數，進而找出每個計算分解基本單位中 iteration 所會使用的陣列資料。在二層迴圈中二維陣列的耦合性陣列下標可以表示成如下的通式：

ArraySymbol[EXP_1][EXP_2]; EXP_i 爲各維的下標函數($1\le i\le 2$)且型式如下:

$EXP_i = a_i*I_1 + b_i*I_2 + c_i$; I_j 爲迴圈索引變數且 a_i、b_i 和 c_i 皆爲常數($1\le j\le 2$)

而仿射陣列存取函數則表示爲 $f_{j,k}(i) = F_{j,k}i+\zeta_{j,k}$，其中 k 表示程序中第 k 個迴圈，i 表示迴圈索

引向量，而 j 則表示迴圈中具資料相依關係陣列的個數。此外，$F_{j,k}i$ 和$\zeta_{j,k}$則分別表示 2*2 的陣列轉換矩陣和 2*1 的位移矩陣。因此，陣列存取函數中陣列轉換矩陣、位移矩陣和陣列下標表示式的關係，則如 3 和 4 所示。

以範例 1 爲例，迴圈中陣列的陣列存取函數則爲:

$$F_{j,k} = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \quad (3)$$

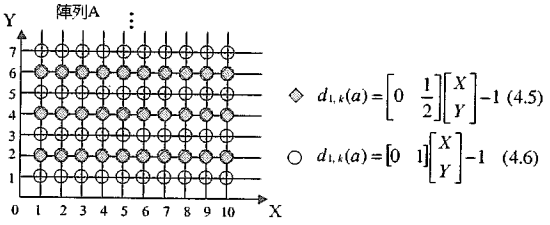
$$\zeta_{j,k} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \quad (4)$$

$$f_{1,k}(i) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} I \\ J \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5)$$

$$f_{2,k}(i) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} I \\ J \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (6)$$

3.2.4 資料分解函數

找出迴圈中陣列的存取函數之後，即可利用陣列存取資訊，找出迴圈中每個計算分解單位中 iteration 使用到的陣列資料，這些資料可由資料分解函數 $d_{j,k}(a) = D_{j,k}a + \delta_{j,k}$來表示。資料分解函數中 k 表示第 k 個巢狀迴圈，j 表陣列中資料分解函數的個數。$D_{j,k}$爲 1*2 矩陣，稱之爲資料分解矩陣。而$\delta_{j,k}$則代表位移常數。



◇ $d_{1,k}(a) = \begin{bmatrix} 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} X \\ Y \end{bmatrix} - 1$ (4.5)

○ $d_{1,k}(a) = \begin{bmatrix} 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \end{bmatrix} - 1$ (4.6)

圖 7 範例 1 資料分解函數及其資料分解情況

由於計算分解函數和陣列存取函數均爲已知，因此藉由 1 和 2，便可求出計算分解函數對每個陣列存取函數所對應的資料分解函數。如範例 1，已知計算分解函數爲 $c_k(i)$=[1 0](i) -1，陣列存取函數如(5)及(6)，並將這些式子代入式子(1)和(2)中，則可得出式子(7)和(8)。圖 7 爲資料分解函數對陣列A 的資料分解，其中灰色菱形和透明圓形分別表示式子 7 和 8 資料分解函數情況。

$$d_{1,k}(a) = \begin{bmatrix} 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} X \\ Y \end{bmatrix} - 1 \quad (7)$$

$$d_{1,k}(a) = \begin{bmatrix} 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \end{bmatrix} - 1 \quad (8)$$

3.2.5 廣域資料分解型態分 (Global data decomposition pattern analyses)

我們的方法是針對資料分解函數中的資料分解矩陣爲對象，其中 $D_{j,k}$爲一 1*2 矩陣，所以可以表示成[a b]。藉由資料分解矩陣的一致，以避免資料重組溝通的花費。而矩陣常數的部份所造成的溝通量可藉由資料分配來減少。因此，本方法的第一原則就是*盡量使程序中每個迴圈的資料分解矩陣一致*。目前我們的對象爲以第一層迴圈或以第二層迴圈爲分解單位。也就是只考慮資料分解矩陣爲[a_1 0]和[0 b_1]的情況，其中 a_1、b_1 代表不爲 0 的實數。

在陣列中資料的放置會依所使用的語言，而有以行爲主(row major)或是以列爲主(column major)的放置方式。而且處理器從記憶體讀取資料放到快取記憶體是以一個區塊爲單位，此區塊的大小決定於快取記憶體線大小(cache line size)。換言之，也就是處理器會一次讀取連續的幾個陣列資料。所以如果在做計算與資料分配時，能夠依據陣列資料放置方式，讓處理器中的計算存取連續的陣列資料，便可增加資料區域性；同時減少快取記憶體讀寫失誤次數，也就是減少記憶體的存取次數。

由於資料分解矩陣爲[a_1 0]表示其資料分解是以第一層迴圈爲單位，意即對陣列是以行爲分解單位，將每一行的陣列資料對應到虛擬處理器空間中。假若計算分解方式，能使資料分解以第一層迴圈爲分解單位，即資料分解矩陣爲[$a_1$0]，則可減少處理器到記憶體存取次數。因此，廣域資料型態分析的第二個原則就是*盡量使資料分解函數中資料分解矩陣爲[a_1 0]*。

但如何將已求出之資料分解矩陣不爲[a_1 0]型態的資料分解函數，如範例 1 之資料分解函數，轉換成[a_1 0]的資料分解矩陣。我們利用不規則相依迴圈中迴圈交換(loop interchange)[11]的方式來達成這個目的。在不規則相依迴圈中迴圈如果能夠滿足定理 1，則表示此兩層迴圈可以交換。

定理 1[11]

第 c 層迴圈可做迴圈交換若且爲若 $V_c(X,Y)*V_{c+1}(X,Y) \geq 0$。其中 $V_c(X,Y)$表示第 c 層迴圈的相依向量。X、Y 爲二個整數變數。

由於在不規則相依迴圈中判斷迴圈是否能交換，是由資料相依關係向量來決定。因此，在開始廣域資料分解型態分析之前，我們由 diophantine equation 找出每個迴圈的的資料相依關係向量，dep_vec_k，其中 k 表示迴圈在程序中的位置。本方法首先檢查程序中每個迴圈的資料分解矩陣是否均是對同一層維度來做資料分解單位，也就是其資料分解矩陣爲[a_1 0]或是[0 b_1]。如果是則表示每個迴圈的資料分解矩陣已達成一致性，此時我們所需處理的便是程序中各個迴圈之間資料分解矩陣的一致性。依據原則二，若程序中資料分解矩陣爲[0 b_1]的迴圈均能夠符合定理 1 做迴圈交換，則爲本方法之最佳狀況。因爲此時程序中迴圈之間的資料矩陣是一致的，且能滿足原則二。若無法求出最佳狀況，則只求符合原則一。若原則一或原則二均無法滿足時，則不要求程序中所有迴圈的資料矩陣均一致。此時，我們逐一對程序中每個迴圈的資料 矩陣和鄰近的資料矩陣加以分析。如果此迴圈的資料矩陣型態和上一及下一迴圈的資料矩陣型態不一致時，檢查此迴圈是否能夠做交換，如果可以便能避免資料重組溝通。

如果程序中迴圈的資料分解矩陣不是對同一層維度來做，則我們便無法使其迴圈間資料矩陣達成一致性。但我們仍可應用原則二的原理，檢查是否有資料矩陣是以第二層維度來做資料分解，如果是則檢查此迴圈是否可以做迴圈交換，若可，則此迴圈的資料分解函數仍能有效利用資料的區域性。最後，根據程序中第一個迴圈之資料分解矩陣加以分析，如果其中有一仿射資料分解函數之資料分解矩陣爲[a_1 0]，則選取此仿射資料分解函數爲最終仿射資料分解函數。否則，便選取第二個仿射資料分解函數爲最終仿射資料分解函數。

在範例 1 中，其資料分解矩陣分別爲[0 1/2]和[0

1]。則此分解矩陣滿足演算法 1 步驟一，並進而在步驟二檢查此迴圈是否能夠做迴圈交換。經由 diophantine equation 計算求出此迴圈的資料相依向量爲(X, Y)。由於 X, Y爲大於 0 的正整數，即 $V_I(X,Y)*V_J(X,Y) \geq 0$，故此迴圈可做迴圈交換。圖 8 即爲範例 1 經廣域資料分解型態分析之後的結果。8(a)爲迴圈交換後之程式碼；由於在步驟二做迴圈交換，所以原本仿射計算分解函數是以第一層迴圈爲分解單位，即計算分解矩陣爲[1 0]。經迴圈交換之後則變成以第二層迴圈爲分解單位，即計算分解矩陣變爲[0 1]，其結果如圖 8(b)所示。此外，原本仿射資料分解函數均是以第二維陣列爲分解單位，其資料分解矩陣分別爲[0 1/2]和[0 1]。經迴圈交換之後，仿射資料分解函數以第一維陣列爲分解單位，其資料分解矩陣則分別變成爲[1/2 0]和[1 0]，其結果如圖 8(c)所示。其中深灰色區塊和淺灰色區塊分別表示放在同一虛擬處理器的計算分解單位以及資料分解單位。

經由廣域資料分解分析之後，程序中迴圈的計算與資料分解函數所對應的虛擬處理器都已決定。但我們發現迴圈中仿射計算分解函數的分解單位，均會對應二個仿射資料分解函數的分解單位。而且不同的計算分解單位會對應同樣的資料分解單位。如圖 8 中以第二層迴圈索引值爲 1 的計算分解單位，會使用到以第一維陣列索引值爲 1,2 的資料分解單位，即深灰色區塊。而二層迴圈索引值爲 2 的計算分解單位，會使用到以第一維陣列索引值爲 2,4 的資料分解單位，即淺灰色區塊。所以此二個計算分解單位均會使用第一維陣列索引值爲 2 的資料分解單位。如果這二個計算分解單位放在不同的處理器中，則表示會有資料溝通的情況產生。因此如何有效地將計算與資料分解單位所對應的虛擬處理器分配到真實機器中，是下一節討論的重點。

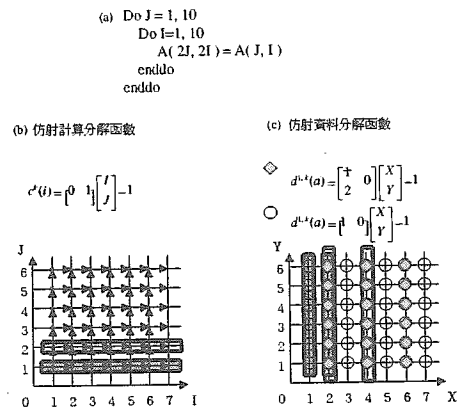


圖 8 範例 1 經廣域資料分解型態分析後之結果。(a)爲迴圈交換後之程式碼；(b)(c)分別爲仿射計算與資料分解函數及其單位方解示意圖

3.2.6 計算與資料分配(computation and data al o-cation)

我們的分配方法是利用區塊或是循環分配的方式，將虛擬處理器分配到真實機器中。至於採用何種分配方式，則是根據迴圈中資料分解矩陣的型態。

根據我們的觀察，當迴圈中相依關係矩陣 $dep_vec_k = (v_1 \quad v_2)$中的 v_1爲一常數且資料分解矩陣 $D_{I,k}$ 是以第一層迴圈爲分解單位，則當系統中處理器的個數爲此常數之因數時，採用循環分配方式會比區塊分配方式較好。因爲若第一層迴圈的相依關係爲常 k 且資料分解矩陣是以第一維陣列資料爲分解單位時，表示第一層迴圈索引值爲 I 的計算分解單位和第一層迴圈索引值爲 I+k 的計算分解單位具有相依關係。同理，第一維陣列索引值爲 x 的資料分解單位和第一維陣列索引值爲 x+k 的資料分解單位具有相依關係。若此時系統中具有處理器個數，以 P 表示，爲 k 值之因數，即 k/P 爲正整數。此時採用循環式分配方式，則第一層迴圈索引值爲 I 的計算分解單位和第一層迴圈索引值爲 I+k 的計算分解單位會分別在第 j 循環和第 j+(k/P)循環放在同一個處理器中；同理具有相依關係之資料分解單位也會放在同一處理器中。

[範例 2]

```
Do I = 1, 10
  Do J = 1, 10
     A( I+3, 2J ) = A( I, J )
  Enddo
Enddo
```

如在在範例 2 中，迴圈的相依關係矩陣爲(3,J)，J 是第二層迴圈索引值。若使用索引同步靜態排程方法時，則迴圈中計算分解函數是以第一層迴圈爲分解單位，如圖 9(a)所示。此時仿射計算分解函數爲[1 0](*i*)-1，而相對應之仿射資料分解函數則如圖 9(b)所示。其中資料分解矩陣分別爲[1 0]和[1 0]。此時，第一層迴圈索引值爲 1 的計算分解單位和第一層迴圈索引值爲 4 的計算分解單位具有相依關係。而第一層迴圈索引值爲 1 的計算分解單位會使用到第一維陣列索引值爲 1,4 之資料分解單位。而第一層迴圈索引值爲 4 的計算分解單位則會使用到第一維陣列索引值爲 4,7 之資料分解單位。但系統中有三個處理器，並應用循環分配方式則第一維陣列索引值 1,4,7 之資料分解單位均會被放置在同一處理器中，如圖 9(b)中，均放在 P0 處理器。其他分解單位也依此類推地分配到系統處理器中。所以，依循環方式來分配計算與資料分解單位會使迴圈中相依關係矩陣 $dep_vec_k = (v_1 \quad v_2)$中 v_1爲一常數之迴圈，避免溝通的情況。

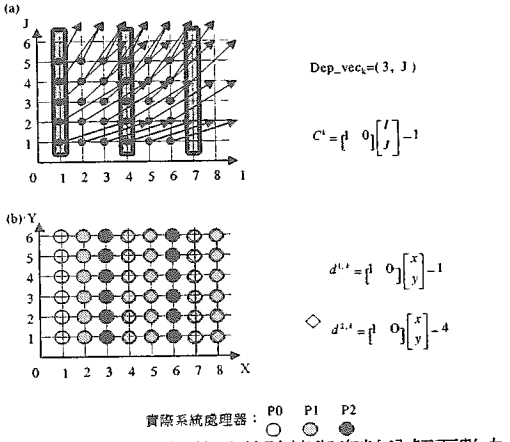


圖 9 範例 2 之相依關係矩陣及其計算與資料分解函數之分配

此外，我們也發現當仿射資料分解函數分別以第一維陣列和第二維陣列爲分解單位，也就是資料分解矩陣分別爲[a_1 0]和[0 b_1]時，若採用循環分配的方式會破壞資料的區域性。因爲，快取記憶體讀取記憶體資料是以快取記憶體線大小爲單位，一次讀入多筆連續的陣列資料。而當資料分解分別爲[a_1 0]和[0 b_1]時，由於陣列資料是以行爲主放置，也就是以第一維陣列爲主放置。因此，快取記憶體會讀取第一維陣列中同一索引值的多筆資料。但仿射資料分解函數中有一資料分解矩陣是以

第二維陣列爲分解單位，造成分配時會破壞快取記憶體讀取第一維陣列時的資料區域性，而其中又以循環分配方式所造成的影響較爲嚴重。

舉例來說，在圖 10 中若有一系統具有三個處理器分別以透明、灰色和黑色來代表；且其仿射計算分解函數，和相對的資料分解函數如圖 10(a)所示。而圖 10(b)(c)則分別表示陣列經區塊分配方式和循環分配方式的結果，其中圓圈和菱形分別代表資料分解矩陣爲[1 0]和[0 1]爲分解單位的資料。而矩形框框則表示快取記憶體一次所讀進的資料大小。在本例中假設一次可讀入 3 個陣列資料單位。由圖 10(b)(c)中可以知道利用區塊分配，快取記憶體每讀一次可以有二次讀取命中。但循環分配方式卻每讀一次，就會產生快取記憶體讀取失誤而必須再至記憶體讀取，無法有效發揮資料的區域性。因此在面對此狀況時則選取區塊分配會有比較好之效果。

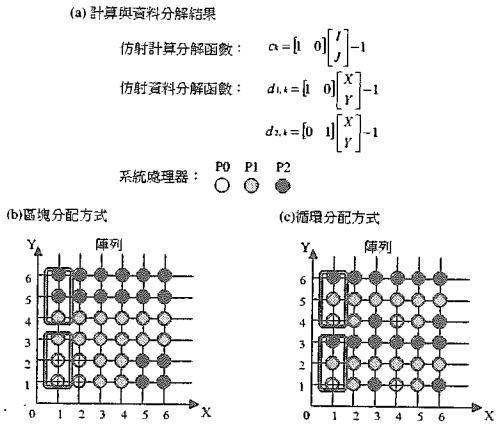


圖 10 快取記憶體線大小和分配方式之關係

目前已經決定出虛擬處理器空間分配到實際機器的方式。接著將計算分解結果和資料分解結果分別放到實際機器的處理器和其區域記憶體中。

由於我們的方法是依據迴圈中靜態排程的方法來決定計算與資料分解函數。同樣地在最後的計算與資料分配的方法上，也必需考量靜態排程的方式。前面我們已經知道迴圈中可平行化區域個數，以 PN_k 來表示。每個可平行化區域可執行的分解單位個數，以 $PRN_k(I)$來表示，其中 I 代表第幾個可平行化區域。以及每個可平行化區域的最小相依向量，以 $MD_{i,k}$來表示。由於每個可平行化區域之間有先後執行的關係。所以如果在做計算與資料分配時，不以可平行化區域爲單位進行分配，而是對整個迴圈進行分配，則可能會使多處理器系統效能無法完全地發揮，因爲具有區域間的相依關係。

由於在計算與資料分解階段，我們已將每個計算分解單位中 iterations 所會使用到的陣列資料以資料分解單位來表示。並將相對應的計算與資料分解單位對應到同一個虛擬處理器中。因此，所謂的計算與資料分配，實際上就是將虛擬處理器分配至實際系統的處理器中。又因相關之計算與陣列資料會對應到同一個虛擬處理器中，所以在同一迴圈中計算與資料分配的演算法是一致的。但由於我們的方法可以應用在廣域分析，所以程序中所有迴圈的計算分解部份均必須加以分配；而不同迴圈中相同陣列資料的分配，則只需分配一次即可。

根據前述之可平行化區域爲分配單位方法，將迴圈中之資料分配從第一個可平行化區域進行分配。將可平行化區域中計算分解單位依區塊或是循環方式加以分配，並將計算分解單位所對應之資料分解單位依同樣之分配方式進行分配，區塊之大小則爲系統中每個處理器可分配到之最大可平行化區域中計算分解單位數。至此不規則相依迴圈已依不規則相依迴圈計算與分配方法將計算與資料分配至處理器及其區域記憶體中。

至此，我們已將原本的不規則相依迴圈，依計算與資料分解函數，將計算與資料分配到處理器中之記憶體。我們不規則相依迴圈計算與資料分配方法，是將目前不規則相依迴圈中靜態排程方法應用在迴圈中，並找出排程所對應的仿射計算分解函數。並利用陣列存取函數找出每個仿射計算分解函數所對應的資料分解函數。接著再依程序中每個對象迴圈中之分解函數進行廣域分析，找出避免資料重組溝通的資料分解型態。最後並依據計算與資料分解函數所對應的虛擬理處理器加以分配至多處理器系統中。

第四章　編譯環境之建立與效益之評估

我們已討論了在不規則相依迴圈中將計算與資料分解並分配到多處理機系統中之過程與方法。利用 SUIF 平行編譯環境與 CONVEX SPP 1000 之分散式共享記憶體多處理器，我們將本論文所提出的方法作實際上的模擬評估。

SUIF (Stanford University Intermediate Format)是一個由 Stanford University 所設計發展的平行編譯系統，這套系統的主要目地是提供做爲高性能計算機平行編譯器之研究平台[14]。

CONVEX Exemplar 系列產品採用可延伸性平行處理技術。CONVEX SPP 基本組成元件爲超節點(hypernode)。一個超節點爲對稱型多處理器 (symmetric multiprocessor，SMP)。每一個超節點具有四組處理器區塊(processor block)，各別含兩顆 PA-RISC 處理器、資料與指令快取記憶體、一個 CPU agent 和一個 CPU 私有記憶體(private memory)。對於 CONVEX SPP 1000，它只具有一個超節點，因此只有八顆有效的處理器。另外，每一個超節點也包含一個或多個超節點私有記憶體，只允許超節點內的處理器存取。記憶體層級(memory hierarchy)定義 CPU 私有記憶體、超節點獨有記憶體、近端分享記憶體(near-shared memory)和遠端分享記憶體(far-shared memory)。 此外，它也提供一個非常人性化的測量工具，稱爲(CONVEX Performance Analyzer，CXpa)，方便使用者掌握執行程式的相關資訊，包含執行時間、cache miss 次數等。

在評估方法上我們一致採用索引同步排程(Index synchronization method)方法[13]，以下以 ISM 來表示，爲不規則相依迴圈計算與資料分配方法中之靜態排程方式。此外，由於 CONVEX SPP 基本組成元件爲超節點(hypernode)，且一個超節點中具有四組處理器區塊，而每塊區塊具有二個處理器、二個快取記憶體和一個區域記憶體。但在 CONVEX SPP 的每個超節點中，每個處理器都有一個區域的快取記憶體，因此我們可以利用將資料讀入快取記憶體的方式來表示資料分配到記憶體的動作。若發生快取記憶體失誤(cache miss)的情況，則表示此時資料並不在處理器本身之區域記憶體中，而必須做遠端記憶體存取。因此，如果快取記憶體失誤量越大，表示遠端記憶體存取次數越多。

我們方法所評比的對象有二。一爲將所有資料放在同一個快取記憶體中，以 ISM_one 表示。另一則爲利用區塊和循環方式[4]將資料做分配，但不考慮相對應之計算分配，以 ISM_block 表示。而我們的方法則以 ISM_Idec 來表示。我們所選用之程式模型爲 ISM[13]方法中之不規則相依迴圈如圖 13(a)(b)所示，分別以 Model1 與 Model2 來表示。

(a) Model1

```
Do I = 1, 500
  Do J = 1, 500
    A(I+J, 3*I+J+3) = ...
    ... = A(I+J+1, I+2*J+4)
  enddo
enddo
```

(b) Model2

```
Do I = 1, 500
  Do J = 1, 500
    A(2*J+3, I+1) = ...
    ... = A(2*I+J+1, I+J+3)
  enddo
enddo
```

圖 13 程式模型

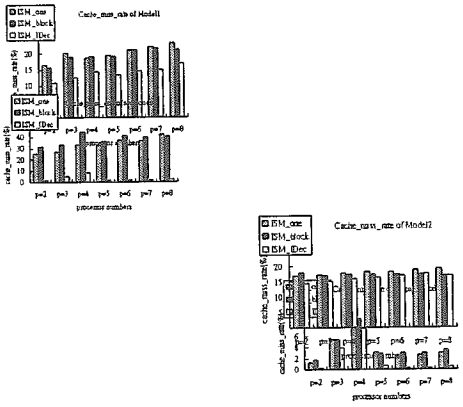
此外，我們也選用在實際應用程式中的不規則相依迴圈爲我們的程式模型。這些程式模型包括 Linpack、Eispack 和 Fishpak 等 Fortran package 中的程式片段。這我們將這些程式模型依其資料相依關係，分別稱之爲 Swap_code1、Swap_code2、Propagate_code1 和 Propagate_code2。其程式碼如圖 14 所示。

4.1 評估結果與分析

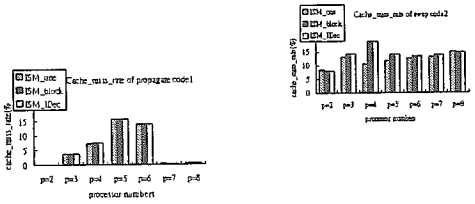
圖 14 實際應用程式之不規則相依迴圈

Swap_code1	Swap_code2	Propagate_code1	Propagate_code2
Do I = 1, Q Do J = 1, R A1 = Y(J,I) Y(J,I) = Y(J,N+1-I) Y(J,N+1-I) = A1 enddo enddo	Do I = 1, Q Do J = 1, R A(I,J) = A(J,I) enddo enddo	Do I = 1, Q Do J = 1, R A(I,J) = B(1,J) enddo enddo	Do I = 1, Q Do J = 1, R A(J,I) = A(J,1) enddo enddo

我們將前一小節中六個不規則相依迴圈， ISM 的靜態排程方式，將迴圈加以排程。其評估結果如下圖所示：



Cache Miss Rate (a) for Model1.(b)for Model2.
(c) for Swap Code 1.(d) for Swap Code 2.



(e) for Propagate Code 1.(f) for Propagate Code 2.

圖 15: 評估結果

如圖 15(a) 所示，對於 model 1 而言，最後我們方法所造成之快取記憶體失誤比率平均要比對資料做區塊分配之 ISM_block 方式少大約 6%，而又比只將資料放在同一快取記憶之 ISM_one 分配方式要少上大約 8%左右。如圖 15(b) 所示，對於 model 2 而言由於我們的分配方式是以雙框矩形爲陣列分配單位，其中包函許多重疊資料，但不若 Model1，對應到這些重疊資料部份之計算分解單位，相隔較遠，即重疊資料所對應之分解單位 I 值差較大，因此無法有效減低資料在不同處理器溝通的次數，所以其所減少的比率大約只有 2%。

此外，在實際應用程式中所使用之不規則相依迴圈部份。圖 15(c) 爲 Swap code1 之結果，這是因爲迴圈做交換後，每個分解單位中 iterations 對資料的存取都以以行爲主的順序存取。因此符合資料之區域性，所以，在 Swap code1 中我們方法減少之快取記憶體失誤比率高達 30%左右。但在 ISM_block 的資料分配方式卻比只將資料放在同一個快取記憶體的情況還差，其原因在於其方法是以[1 0]方向爲分解單位來分配資料，可是此時資料是以[0 1]方向來存取資料，也就是將原本在其他處理器的資料讀到自己的快取記憶體中。但輪到原本分配之處理器要用時，會再發生存取失誤的情形，因此，使得其存取失誤的情況反而比只將陣列資料放在單一快取記憶之情況還差。

在 Swap code2 部份，由於我們的方法也是以[1 0]方向爲資料分解單位，因此，我們的分配方法則和 Block_one 之分配方式一樣。由於每一個計算分解單位中的 iterations 會對應到二個資料分解方向，分別爲[1 0]和[0 1]。而[0 1]會造成如前 Swap code1 中未做迴圈交換之情況，所以我們方法所得到的好處並不多，其結果如圖 15(d)所示。大約能減少 2%的快取記憶體失誤比率。

而在 Propagate code1，其中一個陣列存取函數重覆使用同一個陣列資料分解單位，而此分解單位在我們所評估之三種分配方式中，均會將此塊分解單位分配到同一個快取記憶體中。因此，三種分配之間的差異，只取決於另一個陣列存取函數，而其相對應資料分解是以[1 0]方向爲分解單位。而此分配結果和 ISM_block 方法(ISM_block)之分配結果一致，所以三種分配方法的效果幾乎差不多，如圖 15(e) 所示。

但在 Propagate code2 中，其中一個陣列存取函數也是重覆使用同一個陣列資料分解單位。不同的是此資料分解單位是以[0 1]方向爲分解單位，因而破壞了資料的區域性。而根據我們的方法，我們可以將此迴圈做迴圈交換，使其資料分解單位變成以[1 0]方向爲分解單位，進而維持資料之區域性。而 ISM_block 方法是以[1 0]方向爲分解單位，但資料存取順序是[0 1]方向爲分解單位，就如同 Swap code1 中 ISM_block 分配所造成之情形一樣。因此，其結果反而比只將資料分配到一個快取記憶體的方法還差，如圖 15(f) 所示。。

綜合上述，則我們可以知道，本方法將計算分解單位所對應之資料分解單位加以分配，的確能減少快取記憶體失誤率，如 Model1 和 Model2 所示。但如果陣列中之下標函數不若 Model1 和 Model2 中所示之如此複雜，如在 Swap code1、Swap code2、Propagate code1 及 Propagate code2，下標函數中各層迴圈變數之係數皆不大於 1。則我們之對應資料分解單位分配方法的效果便不明顯。因爲，此時我們的分解方式，常會和只進行資料區塊分配方法之分解方式一致，如 Swap code2 和 Propagate code1。但如果當原本迴圈中 iterations 對資料的存取方式會破壞資料之區域性時，本方法會對此迴圈加以分析比較，觀察其是否可進行迴圈交換，並經由迴圈之交換來發揮資料之區域。因此，如果能夠將我們的分配方法實際應用在資料分配方法上，便能減少遠端記憶體存取次數，進而降低程式執行時間。

第五章 結論與未來研究方向

在多處理器系統發展日益成熟的情況下，良好的平行編譯方法將對整個系統的效益產生重大的影響。而在 NUMA 系統，遠端的(Remote)資料存取時間會比區域性的

(Local)資料存取時間長很多；假如資料放置不恰當，系統會花很多時間在傳遞資料，嚴重影響系統效能。所以，將一個程式的計算(Computation)及資料(Data)適當地分配(Decomposition)到各個處理機上，是平行編譯器的一個重要課題。本論文是針對具有不規則資料相依特性之迴圈做靜態的計算與資料之分配，藉以減少遠端存取之次數。我們並將本論文所提之方法實作在一套平行編譯環境中，並在多處理機系統中實際評估我們的方法。

我們針對只進行迴圈排程而不考慮資料分配方式，以及有做資料分配，但分配方式並不考慮迴圈排程等二種方式爲比較對象，進行模擬評估。可以發現我們的方法大約可以減少 10%左右之快取記憶體存取失誤，也就是能夠減少資料不在區域記憶體情形，降低遠端資料存取次數。因此，如果在實際多處理機系統中，將靜態不規則相依迴圈排程方法，搭配我們的分配方法，應可縮短程式之執行時間。

然而我們的方法仍有幾點值得進一步探討。首先，在目標程式部份，目前我們的方法只針對迴圈中具有相依關係之陣列加以分析。但在實際迴圈中，可能會有多個不同陣列。如果也能將這些陣列加以分析考量，決定適當的分配方式，則可以減少遠端存取之次數。另外，目前我們研究的對象爲二層之迴圈，若能將方法加以推廣至多層迴圈，則可適用的範圍也會越廣。此外，由於目前我們的資料分配方法只適用於分散式共享記憶體多處機系統中，未來也可以將其改進，使其能應用在分散式記憶體多處理機系統中。

NSC 87-2213-E009-049.

參考資料

[1] J. M. Anderson and M. S. Lam, "Global optimizations for parallelism and locality on scalable parallel machines," In Proc. the SIGPLAN '93 Conference on Programming Language Design and Implementation, pp. 112-125. , June 1993.

[2] J. M. Anderson and M. S. Lam, "Data and computation transformations for multiprocessors," In Proc. the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 166-178. July 1995.

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

[4] M. Gupta and P. Banerjee, "Demostration of automatic data partitioning techniques for parallelizing compilers on multicomputers," Transactions on Parallel and Distributed Systems, Vol. 3, No. 2, pp.179-193, March 1992.

[5] C.H.Huang, et.al."Communication-Free Hyperplane Partitioning of Nested Loops," Languages and Compilers for Parallel Computing, p186-200, 1992

[6] C. H. Huang and P. Sadayappan, "Communication-free hyperplane positioning of nested loops, " Journal of Parallel and Distributed Computing, 19(2): pp. 90-102, October 1993.

[7] K.Kennedy, et.al. "Optimizing for parallelism and data locality," In Proc. the 1992 ACM International Conference on Supercomputing, pages 323-334, 1992.

[8] M. S. Lam and M. E. Wolf, "Compilation techniques to achieve parallelism and locality," In Proc. the DARPA Software Technology Conference, pp.150-158, April 1992.

[9] Lenoski. D. E., et al. "The Stanford DASH multiprocessor," IEEE Comput. Vol. 25, No. 3, Mar, 1993, pp.63-79.

[10] Q.Ning, V.V.Dongen and G.R.Gao, "Automatic data and computation decomposition for distributed memory machines," In Proc. the 28th Annual Hawaii International Conference on System Sciences, pages 103-112, 1995.

[11] Derh-Lin Pean, Chao-Chin Wu, Huey-Ting Chua and Cheng Chen. "Effective Parallelization Techniques for Non-uniform Loops", In Proc. the 21st Australian Computer Science Conference, Perth, pp. 393-404, Feb. 1998.

[12] J. Ramanujam and P. Sadayappan, "Compile-time techniques for data distribution in distributed memory machines," IEEE Transactions on Parallel and Distributed Systems, Vol. 2, No. 4, October 1991, pp.472-482.

[13] Ten H. Tzen and Lionel M. Ni, "Dependence Uniformization : A Loop Parallelization Technique," IEEE Trans. On Parallel and Distributed Systems, Vol. 4, No. 5, pp. 547-558, May 1993.

[14] Robert Wilson, et al., An Overview of the SUIF Compiler System, SUIF Manual, Computer Systems Lab Stanford University.1994.

[15] M. E. Wolf. "Improving Locality and Parallelism in Nested Loops, " PhD thesis, Stanford University, August 1992. Published as CSL-TR-92-538.

[16] ftp://ftp.ucar.edu/ftp/dsl/lib/fishpak/

[17] http://elib.zib.de/netlib/eispack/

[18] ftp://ftp.ucar.edu/ftp/dsl/lib/linpack/