Vol.2014-ARC-213 No.18
Vol.2014-HPC-147 No.18
2014/12/9

コンパイラの多重ループ斜め変換による並列性の検出と並列化変換

岩澤　京子[†1]

本発表では，コンパイラが，複雑なデータ依存関係のあるループから，マルチコアや SIMD アーキテクチャに合わせた並列化オブジェクトを生成する手法を記す．２つの傾斜変換を用意して並列化効率の高い方を自動的に選択することが特徴である．一つはよく知られている外側ループインデックスに沿う傾斜化であり，もう一つはこれまで論じられていなかった内側ループに沿う傾斜化である．これらの変換はループインデックスに対する行列操作で定式化することが可能で，ループ本体の実行部分を変更する必要がないため，コンパイラに容易に実装することができる． 2 つの傾斜変換から一方を選ぶ条件や，変換に必要なパラメタをまとめた．最後に，一般には逐次に実行されるバブルソートを内側ループインデックスに沿って並列化する実験を行い，効果を確認した．

Detecting Method of Parallelism from Nested Loops and Parallelizing Conversion by Loop Shearing

KYOKO IWASAWA[†1]

Our loop parallelizing method of compiler for multi-core or SIMD architecture enables parallelized object code to be generated from loops which include complicated data dependency. The characteristic of our method is in choosing the more optimizing method for parallelization from two shearing conversions by inner and outer loop carried data dependences. One of them is novel and involves shearing horizontally along the inner loop index and the other is well-established shearing vertically along the outer loop index. These loop transformations are formalized by matrix operations. They enable the original loop indexes to be expressed using new loop indexes so that compiler does not need to make any change in loop body. At this point, simple templates suffice to generate optimal code. To conclude we summarize the conditions for choosing suitable shearing method and the requirements for conversion.

1. はじめに

近年，マルチコアや SIMD などの様々な並列アーキテクチャを備えた計算機が安価に手に入るようになった．これらのハードウェアを有効に使い，ユーザプログラムの並列性を引き出すコンパイラの処理がますます重要となってきた．

従来，実行比率の高い多重ループを最適化・並列化することによりプログラムの高速化を図る様々な並列化手法が論じられてきた．性能が要求される科学技術計算に現れるループ繰返しにまたがるデータ依存関係が効率のよい並列化変換やベクトル化変換を阻んできた．そこで，本研究では，ループ繰返しにまたがる複雑なデータ依存関係を並列化する方法の一つであるウェーブフロントラインに沿って計算させる手法を取り上げることとした．これは，内側ループに関するデータ依存関係のみならず，外側ループに関するデータ依存関係も解析する必要がある．この変換を施すと，マルチプロセッサのみならず SIMD 型の処理にも同期なしで適用することができる．

本研究の特徴は，ループ交換やループ分割を阻害する複雑はデータ依存関係を含む多重ループの構造を変更することである．とくに，繰返し空間を斜めに歪ませる方向に，内側ループの軸と外側ループの軸の 2 つを用意し，適切な方を選択することである．外側ループを軸とする変換は，よく知られている[4][15]が，内側ループを軸とする傾斜化は論じられてこなかった．しかし，内側ループを軸とする傾斜化も do-all ループを構成して並列化したり SIMD 化するのに有利な場合もある．本論文では，これらの傾斜化の利点と不利な点を示して，コンパイラが最適な並列化変換を行うために一つの方法を選択する方式を記す．

次の 2 章では，2 重ループに対する２つの傾斜化変換による再構成を簡単な例で示す．3 章では，データ依存関係と傾斜角度の関係を記す．4 章では， ２つの変換を一般化した形式を示し，コンパイラのための判断基準とテンプレートにまとめる．5 章では，この方式をバブルソートに適用し，並列実行させた実験結果を示す．最後に 6 章で，結論と今後の進め方をまとめる．

2. 水平方向の傾斜と垂直方向の傾斜

2 種類の傾斜化変換を図 1 の簡単な例で示す．ここで，二重ループの繰返し空間を x-y 座標に取り，内側ループイ

†1 拓殖大学
Takushoku University

ⓒ 2014 Information Processing Society of Japan

ンデックスをx軸, 外側ループインデックスをy軸に取る．そこで，内側ループインデックスに沿う傾斜化を水平方向傾斜変換，外側ループインデックスに沿う傾斜化を垂直方向傾斜変換と呼ぶ．

このプログラムでは，内側ループ繰り越しフロー依存(a)と外側ループ繰り越し逆依存(b)がある．(a)は，内側ループ2回繰り越し，(b)は外側ループ2回繰り越す依存関係である．この2つの依存関係は, ループ繰返し空間では, 図2-(1)のように表される．これらは一つの文で強連結成分となり，並列化もベクトル化も行うことはできない．さらに，(b)のデータ依存関係は，ループ交換も阻害する．本研究の目的は，このようなループから並列実行可能な実行部分を抽出し，効率よく並列化・SIMD 化できるループを再構成することである．

```
float t[NI+2][NJ+2];
for (j=1 ; j<=NJ ; j++) {
  for (i=1 ; i<=NI ; i++) {
    t[i][j]=C1*t[i-1][j+2]+C2*t[i-2][j];
                    (b)
                              (a)
  }
}
```

(a) inner loop carried flow dependence,
(b) outer loop carried anti dependence

図. 1 ループ繰り越し依存関係のあるプログラムの例
Fig.1 Example program and its loop iteration space

図.1 にあるデータ依存関係をループ繰り返し空間で表したものが図 2.-(1)である. 横軸に内側ループインデックス i, 縦軸に外側ループインデックス j をとり，ループ繰り返し本体を並べ，その間のループ繰り返しをまたがるデータ依存関係を矢印でしめしている．

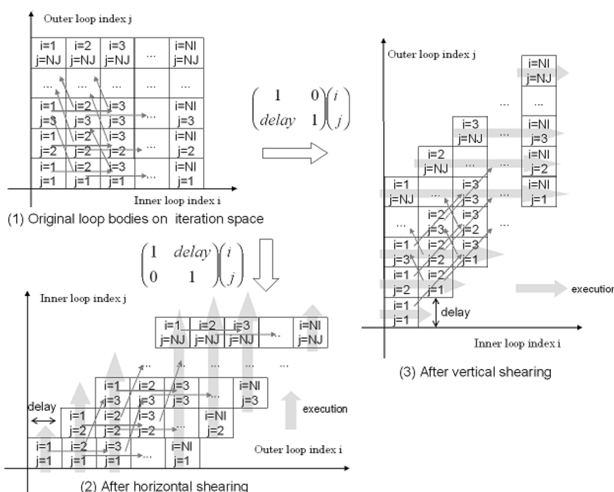


図. 2 ループ繰返し空間での 2 種類のループ傾斜変換
Fig.2 horizontal shearing and vertical shearing on the iteration space

2.1 水平傾斜化（内側ループインデックスに沿う）変換

水平傾斜変換は，データ依存関係を，すべて右（x 軸のプラス方向）を指すようにする．これは，外側ループに関するデータ依存関係の方向を変更するのが目的である．図 1（および図 2-(1)）の例で，(b)の依存関係を右向きになるよう，行列式$\begin{pmatrix}1 & delay\\ 0 & 1\end{pmatrix}\begin{pmatrix}i\\ j\end{pmatrix}$でループインデックスを変換し，図 2-(2)のように，ループ本体の実行順序をずらす．この結果，外側ループのインデックス軸に平行して並ぶループ本体は，同期なしに並列実行可能となり，また，ベクトル化も可能となる．

2.2 垂直傾斜化（外側ループインデックスに沿う）変換

垂直傾斜変換は，でお田依存関係を，すべて上（y 軸のプラス方向）を指すようにする．これは，内側ループのデータ依存関係の歩行を変更するのが目的である．図 1（および図 2-(1)）の例で，(a)の依存関係を上幹になるよう，行列式$\begin{pmatrix}1 & 0\\ delay & 1\end{pmatrix}\begin{pmatrix}i\\ j\end{pmatrix}$で, ループのインデックスを変換し, 図 2-(3)ののように，ループ本体の実行順序をずらす．この結果内側ループのインデックスに平行して並ぶループ本体は，同期なしに並列実行可能となり，またベクトル化も可能となる．

2.3 2 つの傾斜変換の意味

上記２つの傾斜化により，一つのループインデックスの軸に対して並列化やベクトル化をすることができる．図 3 にループ繰返し空間とループ本体を並べた．外側ループインデックス軸 J を縦軸，内側ループインデックス I をよく軸に取っている．これを用いて，(内側ループ i 回，外側ループ j 回)から，ループ繰返しにまたがるデーや依存関係の 3 つの分類を示す．

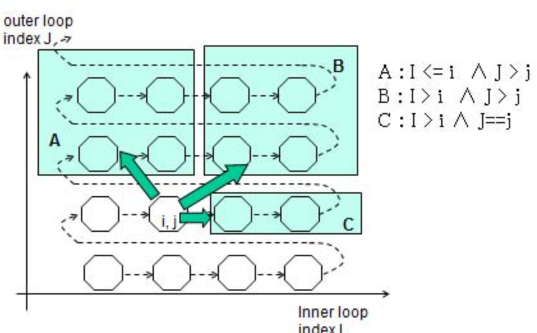


図. 3 ループ繰返し空間でのデータ依存関係
Fig.3 dependences on loop iteration space

ⓒ 2014 Information Processing Society of Japan

(i , j)回目からデータ依存関係が，領域 B にしかなければ，外側ループ，内側ループのいずれの軸についても傾斜変換なしで，do-all 型の並列化をすることができる．領域 A や領域 C に依存関係がある場合は，傾斜化により，依存関係の方向を B 領域に変換して，並列化可能とすることができる．

水平傾斜化は，外側ループの繰り越し依存関係だけの方向を変え，内側ループ依存関係は変更することはない．水平方向傾斜化の利点は，図 2 にある delay の大きささえ十分にとれば，どのようなループにも適用できる点である．最悪な場合としては，内側ループの繰り返し回数を delay に設定すると，逐次実行することになる．もし，2 重ループで一次元配列をアクセスする時のような，領域 A に依存関係があるときに，垂直傾斜化を施すとデータ依存関係が不正となる．

別の観点からみると，垂直傾斜変換は，領域 C への内側ループ依存関係だけ上に向けるため，一般に傾斜化の角度は小さく並列度は高いといえる．次の 3 章で詳細を述べる．

3. データ依存関係と傾斜角度

データ依存関係の制約の下で，並列に実行可能なループ本体を並べるために，データフローを解析してデータ依存関係の距離を計算する必要がある．そして，もっともクリティカルな依存関係を検出し，これが傾斜化の角度を決定する．ここで述べるデータ依存関係の距離やクリティカルな依存関係について，次に記す．

3.1 データ依存関係の距離

すべての配列（変数から並列化したものも含む）について，「定義と参照」「参照と定義」「定義と定義」の組み合わせについて，整数方程式を解いて，依存関係の有無と方向，種類を求める．

配列アクセス(A)から配列アクセス(B)へ，データ依存関係があるというのは，(A)-(内側ループ i 回目，外側ループ j 回目)でアクセスした配列要素を，(B)-(内側ループ i’回目，外側ループ j’回目)でもアクセスされることを意味する．このループ回数によって，次のような関係になる．

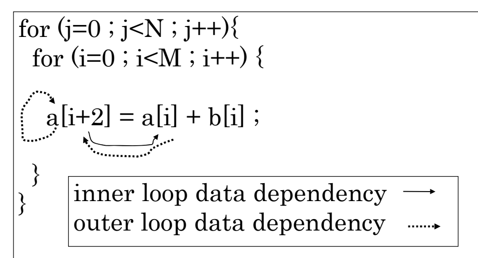
- (j=j’) ∧ (i=i’)：ループの繰返しに関係ないループ独立な依存関係
- (j<j’)：外側ループ繰越し依存関係
- (j=j’) ∧(i<i’)：内側ループ繰越し依存関係

ここで，これらのループ繰返し回数を用いてデータ依存関係の距離を(distO, distI)のペアで定義する．

- distO = j’- j
- distI = i’ – i

一次元配列が 2 重ループの内側で使われる場合は次のようになる．一元配列のアクセス(A)から(B)へ，内側ループの依存関係がある場合は，必ず反対の(B)から(A)へ外側ループ依存関係がある．これは，外側ループの次の回で，再び同じ要素がアクセスされるためである．したがって，内側ループ繰越し依存関係の距離は，(distO=0, distI=i’-i)となり，外側ループ繰越し依存関係の距離は，(distO=1, distI=i-i’)となる．

図 4 に簡単な具体的例を示す．配列 a の定義 a[i+2]から参照 a[i]に対して，内側ループ繰越しフロー依存関係(distO=0, distI=2)があり，また逆に，参照 a[i]から定義 a[i+2]に外側ループ繰越し逆依存関係(distO=1, distI=-2)がある．また，定義に対して，外側ループ繰越し出力依存関係(distO=1, distI=0)がある．



i=i’+2 (0 <= i, i’ < M), is (i,i’)={(0, 2), (1, 3), ... ,(M-3, M-1)}
inner loop carried flow dependency : (distO, distI)=(0, 2)
outer loop carried anti dependency : (distO, distI)=(1, -2)
outer loop carried output dependency : (distO, distI)=(1, 0)

図.4 一次元配列アクセスの依存関係

Fig.4 dependences of one dimensional array access

この例のように，外側ループ繰越し依存関係の距離が，(distO>0 , distI<0)となる場合は，水平方向傾斜化を選択しなければならない．なぜなら，垂直方向に傾斜化を行うと，図 5 に示すように，外側ループの逆依存関係が逆転して，フロー依存関係になってしまうためである．

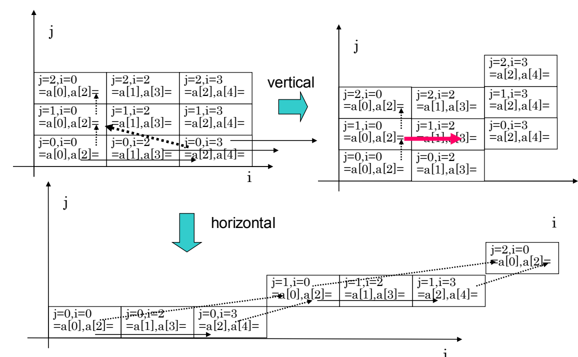


図. 5 図 4 のプログラムの傾斜化変換

Fig.5 shearing of one dimensional array access

ⓒ 2014 Information Processing Society of Japan

3.2 傾斜角度

ループ繰越し依存関係の距離が，図 6 に示すように傾斜化変換の角度を決める．傾斜角度は小さい方が，並列度が増すため，効率よい変換ができる．

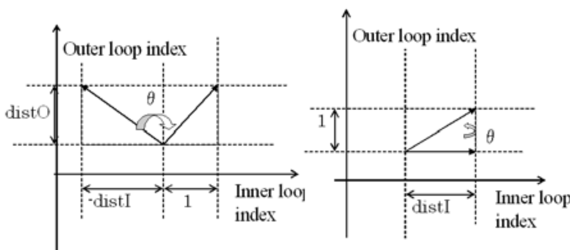
水平方向傾斜化は，すべての外側ループ繰越し依存関係を(distO>0 ∧ distI<0)から(distO>0 ∧ distI'(=1)>0)へ変換する．傾斜角度 θ を最小にするために，distI'は，一番小さい正の整数 1 にする．水平方向でクリティカルな依存関係とは，この θ を最大にする依存関係のことを言う．水平方向傾斜角度は次の式で与えられる．

θ ＝arctan(distI/distO)+arctan(1/distO)

すちょく方向傾斜化は，すべての内側ループ繰越し依存関係を(distO=0 ∧ distI>0)から，(distO'(=1)>0 ∧ distI>0)へ変換する．傾斜角度 θ を最小にするために distO'は，やはり最少の正の整数 1 にする．したがって，傾斜角度は次の式で与えられる．

θ =arctan(1/distI)

この θ を最大にする依存関係を水平方向のクリティカルな依存関係とする．



(1) 水平方向傾斜化　　　　　　(2)垂直方向傾斜化

図. 6 一次元配列アクセスの依存関係

Fig.6 relationship between distance of dependence and shearing angle

3.3 変換行列のパラメタ delay

傾斜変換を行うためのおあらメタとして，2 章に示した，変換行列の delay が必要となる．この値が小さい方か，並列度は増し，効率の良い変換になると考えられる．しかしデータ依存関係を壊さないために，クリティカルな依存関係の距離(distO , distI)を用いる．

$delay = \left\lceil \frac{-distI}{distO} \right\rceil + 1$ を水平方向傾斜変換の行列 $\begin{pmatrix} 1 & delay \\ 0 & 1 \end{pmatrix}$ に用いる．

$delay = \left\lceil \frac{1}{distI} \right\rceil$ を垂直方向傾斜変換の行列 $\begin{pmatrix} 1 & 0 \\ delay & 1 \end{pmatrix}$ に使う．垂直方向傾斜化では，常に delay=1 となるため，実際には，変換行列は $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ となる．

4. 並列化のためのループ傾斜化変換

3 章で定義したいくつかのパラメタを用いて，効果的な並列化変換を行うことができる．本章では，２つの傾斜化変換の方法を述べる．

次の単純 2 重ループを用いて，インデックスの変換を中心に並列化ループを作る．

```
for ( j=0 ; j<NJ ; j++){
    for (i=0 ; i<NI ; i++){
      loop body
   }
}
```

図. 7　傾斜変換する二重ループ

Figure.7 double nested loop before shearing

4.1 垂直傾斜化変換

垂直傾斜化変換は，外側ループインデックスの上(プラス)方向に，データ依存関係をトポロジカルソートして，並列実行可能なループ本体を内側ループインデックスに沿って並べる．オリジナルの二重ループの i-j インデックスを，次の行列によって新しいインデックス I-J に変換する．この時，インデックス I には，同期なしで並列に実行可能なループ本体を並べることができる．

$$\begin{pmatrix} J \\ I \end{pmatrix} = \begin{pmatrix} 1 & delay \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} j + delay * i \\ i \end{pmatrix}$$

この変換によると，外側ループのインデックス J の上限値と下限値が，内側ループインデックス i に依存してしまう. そこで, I と J に関する下限値と上限値から不等式から，外側ループインデックスの上限値と下限値を内側ループのインデックスから独立した式で表現する．

0 ≦ j ＜ NJ → 0 ≦ J-delay*I ＜ NJ
→ delay*I ≦ J ＜ delay*I +NJ
J ≧ delay*I ---(1)　　　J ＜delay*I + NJ ---(2)
0 ≦ i ＜ NI → 0 ≦ I ＜ NI
I ≧ 0 ---(3)　　　I ＜ NI ---(4)

J の下限値：(1)式と(3)式より J ≧ delay *I ∧ I ≧ 0
∴ J ≧ 0
J の上限値：(2)式と(4)式より J＜delay*I + NJ ∧ I＜NI
∴ J ＜ delay*NI + NJ
I の下限値 : (3)式と(2)式より I ≧0 ∧ I ≧(J－NJ)/delay
∴ I ≧ max(0, (J－NJ)/delay)
I の上限値：(4)式と(1)式より I ＜ NI ∧ I ＜ J/delay
∴ I ＜ min(NI, J/delay)

オリジナルのループインデックス i と j は，新しいインデックス I と J を用いて次のように表現できる．

$$\begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} 1 & -delay \\ 0 & 1 \end{pmatrix} \begin{pmatrix} J \\ I \end{pmatrix} = \begin{pmatrix} J - delay * I \\ I \end{pmatrix}$$

したがって，これをループ本体の先頭に挿入することにより，ループ本体の実行文を変更する必要はなくなる．図 7 のプログラムは，図 8 のようになる．この内側 I ループは，同期なしに並列実行することができるし，また，ベクトル化して SIMD 処理することも可能である．

```
for(J=0;J<NJ+delay*NI;J++){
  for(I=max(0,(J-NJ)/delay+1);
    I<min(J/delay+1,NI);I++){  /*parallel*/
      int i=I;
      int j=J-I*delay;
      loop body
   }
}
```

図. 8　垂直方向（外側ループ）傾斜変換した二重ループ
Figure.8 vertically sheared loop

4.2 水平傾斜化変換

水平傾斜化変換は，内側ループインデックスの右（プラス）方向に，データ依存関係をとぽロジカルソートして，並列実行可能なループ本体を外側ループインデックスの軸に沿って並べる．オリジナルのループの二重ループのインデックス i-j を，次の行列によって新しいインデックス I-J に変換する．この時インデックス J には，同期なしで並列に実行可能なループ本体を並べることができる．

$$\begin{pmatrix} J \\ I \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ delay & 1 \end{pmatrix} \begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} j \\ delay * j + i \end{pmatrix}$$

オリジナルのループインデックスは，次の行列式で表されるので，これをループの本体の先頭に挿入し，次の図 9 のようなプログラムとなる．

$$\begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -delay & 1 \end{pmatrix} \begin{pmatrix} J \\ I \end{pmatrix} = \begin{pmatrix} J \\ -delay * J + I \end{pmatrix}$$

```
for (J=0;J<NJ;J++) {
  for(I=J*delay; I<J*delay+NI; I++) {
      int i=I-J*delay;
       int j=J;
      loop body
   }
}
```

図. 9　水平方向（内側ループ）傾斜変換した二重ループ
Figure.9 horizontally sheared loop

しかし，ループインデックス I の方向には，依存関係があり，逐次に実行する必要があるため，このまま，外側ループの J で並列化することはできない．そこで，ループ交換により並列実行可能なループを内側にする．ループ交換を阻害するデータ依存関係は，すべて傾斜化により，阻害要因ではなくなっている．

ただし，交換により外側なるループインデックス I の上限値や下限値は，内側になるループインデックス J に依存するため，水平方向傾斜化と同様に外側ループインデックスを，不等式を用いて変換する．

$0 \leqq j < NJ \rightarrow 0 \leqq J < NJ$
$J \geqq 0$ ---(1)　　　$J < NJ$ ---(2)
$0 \leqq i < NI \rightarrow 0 \leqq I-delay*J < NI$
$\rightarrow delay*J \leqq I < NI+delay*J$
$I \geqq delay*J$ ---(3)　　$I < NI+delay*J$ ---(4)

I の下限値：(3)式と(1)式より $I \geqq delay*J \wedge J \geqq 0$
$\therefore I \geqq 0$
I の上限値：(4)式と(2)式より $I < delay*J + NI \wedge J < NJ$
$\therefore I < delay*NJ + NI$
J の下限値：(1)式と(4)式より $J \geqq 0 \wedge J \geqq (I-NI)/delay$
$\therefore J \geqq \max(0, (I-NI)/delay)$
J の上限値：(2)式と(3)式より $J < NJ \wedge J < I/delay$
$\therefore J < \min(NJ, I/delay)$

この変換により，次の図 10 のような目的とするループを作成することができる．この時の内側 J ループは，同期なしで各繰返しを並列に実行することも，ベクトル化することも可能である．

```
for(I=0; I<NI+delay*NJ); I++) {
  for(J=max(0,(I-NI)/delay+1);
   J<min(I/delay+1,NJ);J++){/* parallel */
      int i=I-J*delay;
      int j=J;
      loop body
   }
  }
```

図. 10　ループ交換し並列化したループ
Figure.10 exchanged loop

4.3 コンパイラの変換のためのまとめ

これらの変換は，複雑そう見えるが，コンパイラがこの変換過程をすべて行う必要はない．4.1 節で述べた垂直方向傾斜化や 4.2 節で述べた水平方向傾斜化を行うためには，次のパラメタを計算可能なオブジェクトコードを生成すればよい．

(1) NI：内側ループの繰返し回数
(2) NJ：外側ループの繰返し回数
(3) distO：外側ループにクリティカルがデータ依存関係の距離
(4) distI：内側ループにクリティカルなデータ依存関係の距離

ⓒ 2014 Information Processing Society of Japan

(5) delay： $delay = \left\lceil \frac{-distI}{distO} \right\rceil + 1 \quad delay = \left\lceil \frac{1}{distI} \right\rceil$

傾斜化変換のパラメタ

並列化のための傾斜化変換を行うために．コンパイラはループ子繰越し依存関係の距離を計算すればよい．オリジナルのループインデックスを，新しい変換後のループインデックスで表した代入文をループ内部に挿入すれば，ループ本体の実行文を変更する必要はない．コンパ入り処理のソースの形式が残っている早い時期の本変換を適用するのが望ましい．

二重ループの場合，2 種類あるどちらの傾斜化を適用するかが常用である．その原則を次の 2 つにまとめた．

(Case1) 外側ループ繰越し依存が存在し，その距離が (distO>0, distI<0)となるな場合は，水平方向傾斜化（内側ルーインデックスに沿う傾斜化)を選択しなければならない．

(Case2) Case1 以外では，変換後の外側ループの繰返し回数が短くなる変換を選ぶ．内側ループの繰返し回数は並列度を表すが，外側ループは逐次に実行しなければならず，並列化の効率を決めるためである．すなわち，条件式

NJ+delay*NI　> NJ+ NI

が成立する場合は，垂直方向傾斜化（外側ループインデックスに沿う）を選び，それ以外の場合は，水平方向傾斜化を選ぶことになる. delay は, 常に 1 より大きいため, (Case1) の場合以外は，垂直方向傾斜化を選ぶ.

5. バブルソートを用いた実験

4 章に記した原則に従って，一次元の配列を二重ループで用いるバブルソートのプログラムを並列化し，実行時間(レスポンスタイム)を測定した．

図 11 に利用したバブルソートプログラムと変換したプログラムを載せる．要素の入れ替えの際に，内側ループについて次の回に繰り越すフロー依存関係と逆依存関係があり，どちらも距離は，(distO=0, distI=1)である．また，一次元の配列を用いているため外側ループ繰越しの依存関係もいくつかあるが, クリティカルなものは, (distO=1, distI=-1) となるフロー依存関係である．この関係より，垂直傾斜化を行うことができないため，delay=(-1/1)+1=2 を持ちって，水平方向傾斜化（内側ループに沿う）を行う．新しいループインデックス p と q を用いて，図のようにプログラムを変換すると，内側ループ p の各繰返しについて独立して並列に実行することができる．この変換をループ繰り返し空間で示すと，図 12 のようになる．水平方向に傾斜化したためループ交換が必要となり，内側ループ p の沿って並ぶ繰り返し本体部分は，並列実行も SIMD 実行も可能となる．

図 11.-(2)プログラムの実行時間を測定したものを図 12 に示す．これは，インテルの Core　4 コアを用いた．また，OpenMP の指示文を入れたソースプログラムを生成し，インテルの OpenMP コンパイラで実行した結果である．

十分にデータ量が与えられると，一般に逐次的な処理と考えられるバブルソートも，4 並列のマルチコアで，約 2 倍の性能向上となっているのがわかる．これは，delay が 2 であるため，オリジナルプログラムに比べ逐次処理の外側ループ長が 2 倍になっているためである．

```
void BubbleSort(double A[], int N)  {
 for (int j=0; j<N-1; j++) {
  for (int i=0; i<N-j; i++){
     if (A[i]>[A[i+1])
       {  double w=A[i];
          A[i]=A[i+1];
          A[i+1]= w ;  }
    }
   }
}
```

(1) バブルソートプログラム

```
void BubbleSort(double A[], int N)  {
 for (int q=0; q<2*(N-1); q++) {
   int initial=max(0,q-(N-1);
   int last = min(q/2, N-1);
  for (int p=initial; p<last; p++){
    i=q-2*p;    if (A[i]>[A[i+1])
     {  double w=A[i];
        A[i]=A[i+1];
        A[i+1]= w ;  }
   }
  }
}
```

(2) 水平方向傾斜化により並列バブルソートプログラム
delay=2 (dist0=1, distI= -1)

図. 11　水平傾斜化により並列化したバブルソートプログラム

Figure.11parallelized bubble sort program by shearing

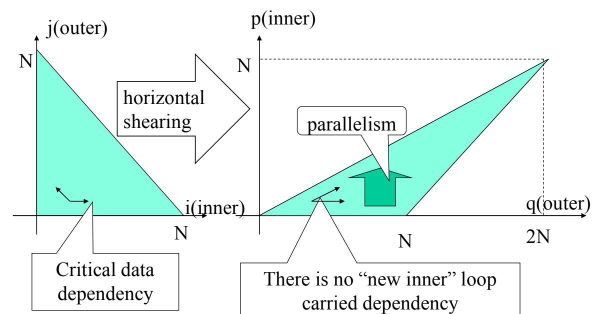


図. 12　繰返し空間でのバブルソートを水平傾斜化

Figure.12 horizontally sheared bubble sort on iteration space

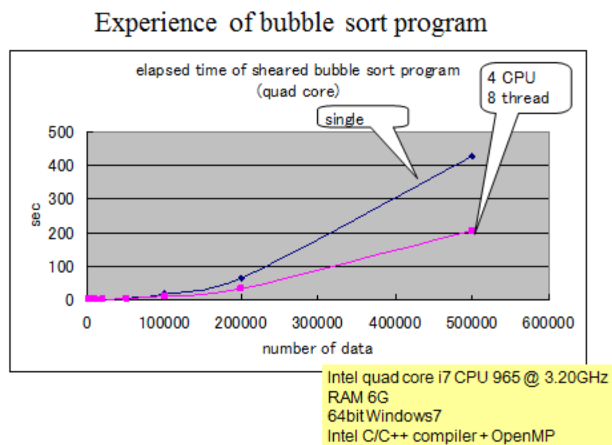


図 13　傾斜化により並列化したプログラムの実行効率
Figure.13 efficiency of parallelized bubble sort program by shearing

6. 関連研究

ループの並列化に関しては多くの研究があり，配列のデータフロー解析は広範囲にわたり研究されてきた．Kuck ら[1]の論文は並列化・ベクトル化コンパイラに影響を与え，配列要素の依存テスト[2]も多く使われてきた．データ依存解析や並列化のためのコンパイラの技術は[3][4]に総括されている．3 章の解析は，それらの技術に基づいているが，それらは最内側ループの解析のみを論じている．Omega プロジェクト[5][6][7]は，配列化コンパイラのための配列データフロー解析をより一般化し詳細化した．外側ループに関する配列のデーやフロー解析は，[8][9]に論じられている．

Lamport[10]は，並列に実行するためのハイパープレーン方式の原理を示した．われわれの方式もまた，多重ループにおけるウェーブフロントラインに沿う並列性を検出している．[11],[12]では，メモリ共有型のマルチプロセッサにおいて DOACROSS 型のループ並列化を行うときの同期処理やメッセージパッシングの最適化や並列処理の効率を論じている．

[15]や[16]では．ウェーブフロントに沿う並列化を紹介している．これらは，本報告に示した垂直方向(外側ループインデックス)傾斜化について論じられており，ループ交換を必要とする水平方向(内側ループインデックス)方向の傾斜化には，触れていない．われわれの研究がこれらと異なるのは，2 種類の傾斜化変換を用意し，与えられたソースプログラムに合わせて，並列公立のよい方を選択することである．これにより適用範囲も広がり，また，これらの変換をパタン化しテンプレートとして用意することにより，必要なデータ依存関係が与えられれば，コンパイラが自動的にプログラム変換することができる．

7. おわりに

コンパイラが行う 2 重ループの自動並列化方式を示した．これは，複雑なループ繰り越し依存関係があるため，そのままでは並列化，ベクトル化，さらにはループ交換やループ分割もできないような場合にも適用できる可能性がある．この方式の特徴は，用意してある 2 つの傾斜変換から効率が良いと判定できるものを選ぶことである．水平方向（内側ループインデックスに沿う）傾斜化は，これまで論じられて来なかった手法である．一般には，水平方向（外側ループインデックスに沿う）傾斜化より並列化効率は悪いが，適用範囲は広く，正確にデータフローを解析すればどのような場合でも適用することができる．ただし，最悪の場合は，逐次に実行することになる．

水平傾斜化の適用例として，5 章にバブルソートの並列化を行い，十分多くのデータを処理する場合は，効果を上げることができることが分かった．

これらの変換の過程は複雑に見えるが，ループインデックスに対して行列演算を行うことによって実現できるため，ループ本体の実行文を変更する必要がない．そこで，変換に必要なパラメタを計算し，効率の良い傾斜変換の選択基準を示した．また，ループの構造をテンプレート化することにより，コンパイラに取り入れて自動的に変換することができる．

今後の課題として，今回は対象を 2 重ループとしたが，より一般化した多重ループや，ループ本体のブロック化による並列性の検出を検討することがあげられる．そのためには，“polyhedral analysis” [13][14]を取り入れることにより，この手法もより適用範囲が広くなると考えらえる．

参考文献

[1] Kuck, D. J., Kuhn, R. H., Padua, D. A., Leasure, B., Wolfe, M. Dependence graphs and compiler optimizations, Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages pp.207 – pp.218 (1981)

[2] Banerjee, U. K., Dependence Analysis for Supercomputing, Kluwer Academic, (1988)

[3]. Zima, H and Chapman, B. Supercompilers for Parallel and Vector Computers. Addison-Wesley, (1990)

[4] Randy, A. and Kennedy, K., Optimizing Compilers for Modern Architectures. Morgan Kaufmann, (2001)

[5] Pugh, W. and Wonnacott D., Constraint-Based Array Dependence Analysis, ACM Trans. of Programming Languages and Systems, Vol. 20, No. 3, pp.635-678. (1998)

[6] Maslov, V., Lazy Array Data-Flow Dependence Analysis, ACM Proceedings of POPL, pp.31-325 (1994)

[7] Pugh, W., The Omega Test: a fast and practical integer programming algorithm for dependence analysis, ACM Proceedings of Supercomputing’91 pp.4-13. (1991)

[8] Maydan, D. E., Amarasinghe, S. P., Lam, M. S., Data Dependence and Data-Flow Analysis of Arrays, Proc. of 5th Workshop on Languages and Compilers for Parallel Computing, pp.434-448 (1992).

ⓒ 2014 Information Processing Society of Japan

[9] Maydan, D. E., Hennessy, J. L., Lam, M. S., Effectiveness of Data Dependence Analysis, International Journal of Parallel Programming, pp.63-81,(1995).
[10] Lamport,L., The Parallel Execution of Do Loops" Communications of the ACM, Vol.17, No2, (1974) pp.83-93
[11] Rajamony, R., Cox,A.L., Optimally Synchronizing DOACROSS Loops on Shared Memory, International Conference on Parallel Architecture and Compilation Techniques (1997)
[12] Hong-Men Su, Pen-Chung Yew, Efficient Doacross Execution on Distributed Shared-Memory Multiprocessors, ACM, Proceedings of the 1991 ACM/IEEE conference on Supercomputing, pp.842-853, (1991).
[13] Vasilache, N., Bastoul, C., Cohen, A., Polyhedral Code Generation in the Real World, proceedings of 15th International Conference CC2006, pp.185-201, (2006)
[14] Srikant, Y.N., Shankar, P., The Compiler Design Handbook, CRC Press, (2003).
[15] Wolfe, M., Loop Skewing: The Wavefront Method Revisited, International Journal of Parallel Programming, Springer Netherlands, pp.279-293, (1986)
[16] Kim, K., Nicolau, A., Parallelizing tightly nested loops, Proceedings of Parallel Processing Symposium, pp.630-633(1991)

ⓒ 2014 Information Processing Society of Japan