可視化されたデータ依存の分類とその並列化手法への適用

山口智美[†] 笹倉万里子[††] 城和貴[†]

自動並列化コンパイラの研究では，長い間，ループ変換理論やデータ依存解析の理論的な研究が行われてきた．一方，並列システムは共有メモリ，分散メモリ，クラスタ，Grid と様々なトレンドが出現しており，自動並列化コンパイラがすべてに実装されるのを期待することは現実的ではない．現に多くの研究者が各々のアプリケーションの並列最適化を行うのに，MPI，OpenMP，HPF 等の環境を使っている．ところが並列プログラミング初学者には，ループ変換理論やデータ依存解析を直接利用するには敷居が高く，自動並列化コンパイラ研究で培われてきた研究成果を利用できないという問題点があった．本論文では，並列プログラミング初学者がデータ依存関係を目視により直観的に理解し，得られたパターンからループ変換手法を対話的に適用する手法について提案し，その具体例について報告する．

Classification of Visualized Data Dependence and Its Application to Parallelization

Tomomi Yamaguchi,[†] Mariko Sasakura[††] and Kazuki Joe[†]

Recently, as the use of parallel computers becomes popular in the field of scientific calculation, effective converting methods from conventional sequential programs into corresponding parallel ones are required. For effective conversions, however, vast amounts of skills and knowledge for parallel programming are needed for programmers. In this paper, we propose the introducion of a 3D visualization system called NaraView as a programming environment for the effective parallelization. NaraView analyzes data dependences in a given program code, and visualizes their relations. By the visualization of program codes, we can recognize typical patterns in the shape of visualized data dependences, and implement corresponding parallelizing methods easily. We show effective parallelizing methods which have such typical patterns in loop structures of concrete program codes.

1. はじめに

グリッドや PC クラスタ，並列計算機等において大規模な科学技術計算を行う際に，プログラムの並列化は必要不可欠なものである．一般に，並列処理環境や並列システムの研究開発に従事する研究者は，当該科学技術分野に関する知識に乏しく，逆に各科学技術分野の研究者は，自分のアプリケーションに適した並列化がどのようなものであるかという知見が少ない．

並列化支援視覚化[☆]システム NaraView[2] はそのような問題を解決することを目的として開発されたものであり，並列化の指針を視覚的直観的に与える．しかしながら，NaraView によって与えられた並列化支援の可視化結果のみでは，適切な並列化手法をアプリケーション・ユーザが発掘するには不十分である．また，アプリケーションを理解することなく並列化を行うのは，同様に可視化結果が与えられても，並列化の専門家ですら，時として困難である．

我々は NaraView の効果的な利用方法を探る[1] うちに，NaraView が生成する可視化結果にある種のパターンが存在し，可視化されたデータ依存[3] の形状と並列化手法の関連付が可能であるという知見を得るに至った．本論文では，並列化を行う際に基本的な技術であるループ変換[3] が適用可能なデータ依存関係を NaraView で可視化を行い，パターン化を試みる．

本論文の構成は以下のとおりである．2 章では NaraView の概要について述べ，3 章では可視化された

† 奈良女子大学
Nara Women's University
†† 岡山大学
Okayama University

☆ NaraView はこれまで視覚化という表現を用いていたが，これはサイエンティフィック・ビジュアリゼーションで用いられる可視化との区別を行うためであった．その後，可視化という表現を用いても誤解が生じないという確信が持てたため，本論文では視覚化という表現ではなく，可視化という表現を用いる．

データ依存の分類に関する基本的なアイデアを示し，4 章では具体的にループ変換手法と可視化されたデータ依存の関連付を行う．また，5 章では並列化の可視化支援に関する関連研究について概説する．

2. NaraView の概要

NaraView はもともと自動並列化コンパイラを開発する際の視覚的支援システムとして開発が行われた[4]．現在は，我々が開発を進めている自動並列化コンパイラ PROMIS-NWU [5] に組み込まれた形で実装されている．NaraView は与えられたプログラムの情報を表す 3 種類のビューで構成される．

- プログラム構造ビュー（**PSV**）：
  プログラム全体をツリー構造で表す．
- ソースコード・ビュー（**SCV**）：
  PSV でユーザにより指定された部分のプログラムのソースコードを表示する．
- データ依存ビュー（**DDV**）：
  PSV でユーザにより指定された部分のデータ依存を表現する．

これらのビューを用いて，ユーザは与えられたプログラムを並列化するのに最適な戦略を決定する．以下に NaraView を用いて並列化を行う際の典型的な手順を示す．

(1) ユーザが与えられたプログラムの全体構造を直観的に把握できるように **PSV** を表示する．
(2) 着目すべきループを選ぶ．
(3) 選ばれたループの **SCV** と **DDV** を表示する．
(4) 効果的な最適化手法やループ変換を選ぶ．

2.1 データ依存ビュー

データ依存はループを並列化できるかどうかを検討するのに最も重要な情報の 1 つである．データ依存ビュー（**DDV**）は条件分岐文を内部に含まない任意のループに対して適用され，データ依存状態を可視化する．**図 1** に **DDV** の例を示す．**DDV** では，選ばれたループ内で参照される配列要素ならびにスカラ変数が xy 平面上に投影され，その z 軸方面でプログラム・フローを表す．xy 平面での各点は配列の要素もしくはスカラ変数が割り当てられ，z 軸の単位はオプション指定により，ステートメントもしくはベーシック・ブロックが選択可能である．

各配列や変数の xy 平面への投影は次のように行われる．注目しているループ内で最初にアクセスされる配列もしくは変数は原点から投影され，それがスカラ変数の場合には，xy 平面の $(1,0)$ から次の配列もしくは変数の投影が行われる．1 次元配列の場合は，y 軸方向に当該 1 次元配列が投影され，xy 平面の $(1,0)$ から次の配列もしくは変数の投影が行われる．2 次元配列の場合には，xy 平面に投影が行われ，xy 平面の $(m,0)$ から次の配列もしくは変数の投影が行われる．ただし，m は 2 次元配列の x 軸方向の長さである．3 次元以上の多次元配列の場合には，オプションで選んだ 2 次元部分のみが投影される．

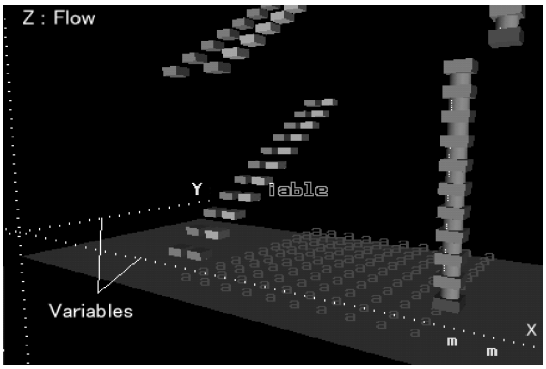

図 1 データ依存ビュー
Fig. 1 Data dependence view.

配列要素やスカラ変数が参照される場合，該当する xy 座標とプログラム・フローでの参照時間（z 軸）にキューブ（立方体）として表示される．

各立方体は色の違いにより 3 種類の参照を表現する．

- 青色：リード
- 赤色：ライト
- 紫色：リード&ライト

同じ xy 座標上で z 軸に沿って複数のキューブが存在する場合，それは同じデータに対する複数参照を意味し，データ依存関係の存在を表現する．**DDV** ではこのような場合，複数のキューブはポール（円柱）で結ばれ，データ依存の存在を直観的に分かりやすくしている．ポールには次の 2 種類がある．

- ライフタイム・ポール：フロー依存[3] の存在を示す緑色のポール
- 逆依存ポール：逆依存[3] の存在を示す黄色のポール

3. 基本的なアイデア

本章では，**DDV** を利用する際の基本的なアイデアについて説明する．まず，xz 平面や yz 平面にしかキューブが存在しないなら，多くの場合，それは単一な（多重ではない）ループである．多重ループでは，多くの場合，キューブは xyz 空間に 3 次元的に観測される．

ループが単一であるとき，そのループ内で参照され

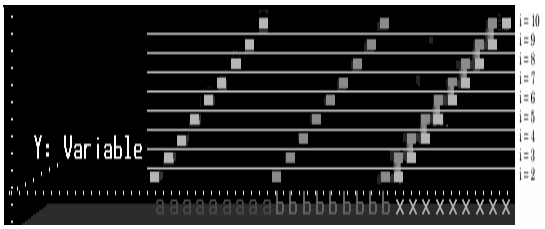

図 2 単一ループ
Fig. 2 A single loop.

```
DO 100 i = 2, 10
 S1:      a(i) = a(i) + b(i)
 S2:      x(i) = x(i) + x(i-1) + a(i)
 100 CONTINUE
```

図 3 図 2 のソースコード
Fig. 3 Source code of Fig. 2.

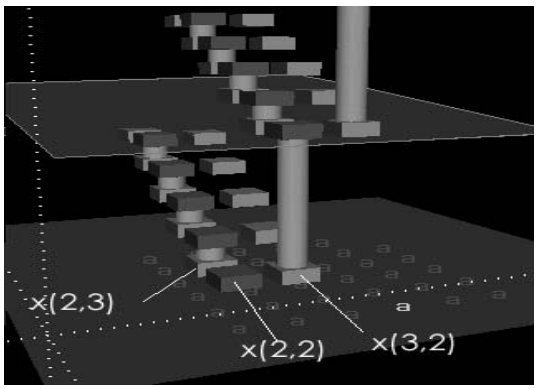

図 4 二重ループ
Fig. 4 A double loop.

る配列（ベクトル）に対応したキューブは x 軸（もしくは y 軸）に沿って現れ，z 軸はイタレーションの実行順序を表現する．**図 3** は一次元配列（ベクトル）を参照する単一ループ例であり，そのデータ参照の可視化結果は**図 2** で示されている．図 2 ではキューブはイタレーションの実行順序に従い上方向に配置されている．$(i, x(i))$ と $(i-1, x(i))$ の間にあるポールはイタレーション $i-1$ と i の間でのデータ依存を表す，クロス・イタレーション（ループ繰越し）の依存である．

多重ループの場合，そのループ中で参照されるデータを表すキューブは xy 平面に沿って配置され，イタレーションの実行順序は z 軸で表される．たとえば**図 5** のプログラムの **DDV** を示す**図 4** において，$(2,2)(2,3)(3,2)$ のキューブは $i=2$，$j=2$ のイタレーションで参照されることを意味し，同一 xy 平面上に配置されている．それらを起点として j をインデックスとするイタレーションに従って上方向に配置されているキューブは同一ループ（インデックス j）でのアクセスを意味している．図 4 では長さの異なる 2 種類のポールが観測されるが，短い方は内側のループでの依存を，長い方は外側のループでの依存を表している．

```
DO 100 i = 2, 6
       DO 200 j = 2, 6
         a(i,j)=a(i+1,j)+a(i,j-1)
 200   CONTINUE
 100 CONTINUE
```

図 5 図 4 のソースコード
Fig. 5 Source code of Fig. 4.

4. データ依存の目視による分類

ループ中で参照される配列要素やスカラ変数のデータ依存を調べつつ，そのデータ依存関係を保ったまま適切なループ変換を適用するのは至難の技である．NaraView を利用した並列化では，ビューを利用して対話的に最適化手法を探るのであるが，本論文では最適化手法として自動並列化コンパイラで利用されるループ変換[6),7)] について議論する．与えられたプログラム中の各ループの配列参照を可視化し，そのパターンから有効で適切なループ変換手法を決定する．ただし，対象とするループ変換は，ループ分配，ループ交換，ループ傾斜，ループ細分，ループ皮むき☆である．

4.1 ループ分配

ループ分配は複数の文をループ・ボディに含む 1 つのループを，セマンティクスを変えることなく複数のループに分配する手法である．図 2 は図 3 のループ中の変数参照を可視化したものである．図右側の $x(i)$ と $x(i-1)$ の間にフロー依存を示すライフタイム・ポールが観測される．このポールはイタレーションをまたいでいるので，ループ繰越データ依存であり，このまま DOALL 型の並列化はできない．

図 6 は図 3 にループ分配を適用して 1 つのループを 2 つのループに分割した後のソースコードである．また，**図 7** はそれぞれのループの変数参照を可視化したものである．図 7 上は文 S1 を含むループでの変数参照を表しており，データ依存を表すポールがないことから，このループに関しては DOALL 型の並列化が可能なことが分かる．一方，図 7 下は文 S2 を含むループでの変数参照を表しており，依然としてデー

☆ Loop distribution, Loop interchange, Loop skewing, Strip mining, Loop peeling の文献 3) による日本語表記

```
DO 100 i = 2, 10
 S1:     a(i) = a(i) + b(i)
 100 CONTINUE
     DO 200 i = 2, 10
 S2:     x(i) = x(i) + x(i-1) + a(i)
 200 CONTINUE
```

図 6 ループ分配後のソースコード
Fig. 6 Source code of the loop distribution result.

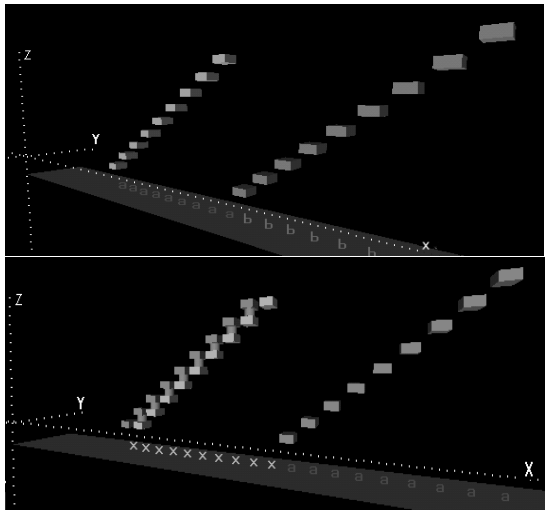

図 7 ループ分配後の配列参照の可視化
Fig. 7 Visualization of the array references with the loop distribution.

タ依存を示すライフタイム・ポールが存在することから，このループは DOALL 型並列化はできないことが分かる．このように，ループ分配を適用した結果，元のプログラムの半分は並列化可能となった．

- **ループ分配パターン：**あるループの **DDV** がイタレーション間にポールを持つとき，ポールの配列参照部分と，それ以外の（配列を含む）変数参照部分にループを分割することができる．

通常，ループ分配を適用するには，巡回依存がある場合は変数のリネーミング等である程度データ依存を解消してから，ループ・ボディの文の再配置を経て最適な形で適用する．本パターンでは，そこまでは適用できないが，巡回依存のない場合に関しては，ループ分配が可能であることをユーザに示すことができる．

なお，本パターンは，多重ループの場合でも，内側のループを表示させないことで適用可能である．

4.2 ループ交換

ループ交換は多重ループにおいて，内側のループと外側のループを入れ替える手法である．**図 8** と**図 9** はループ交換の適用前と後の例である．また，それぞれの配列参照可視化結果を**図 10** に示す．

```
DO 100 i = 2, 5
        DO 200 j = 2, 10
          b(i,j) = b(i+1,j) + a(i,j)
 200    CONTINUE
 100 CONTINUE
```

図 8 ループ交換前のソースコード
Fig. 8 Source code without the loop distribution.

```
DO 100 j = 2, 10
        DO 200 i = 2, 5
          b(i,j) = b(i+1,j) + a(i,j)
 200    CONTINUE
 100 CONTINUE
```

図 9 ループ交換後のソースコード
Fig. 9 Source code with the loop distribution.

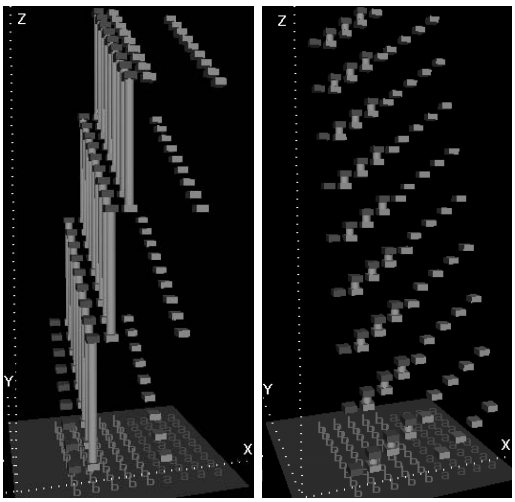

図 10 ループ交換前（左）と後（右）の **DDV**
Fig. 10 **DDV** with the loop interchange (left) and without the loop interchange (right).

図 10 左では，配列 b に起因する逆依存ポールがあるにもかかわらず☆，ループ i の各イタレーションごとに，ループ j で DOALL 型並列化が可能である．

j は内側のループであり，一般に多重ループにおける内側のループ並列化より外側の並列化の方が効率が良い[3] ため，ループ交換を試みる．実際，ループ交換を適用した後の図 10 右では j に関するループ繰越データ依存は存在せず☆☆，DOALL 型並列化が可能であることが分かる．

- **ループ交換パターン：**多重ループの **DDV** において，ループ繰越しのポールを持つループと持たな

☆ j に関するループ・イタレーションを表示しなければ，i に関するループ繰越データ依存が容易に観測できる．
☆☆ ループ i を非表示にすれば容易に分かる．

```
DO 100 i = 2, 6
       DO 200 j = 2, 6
         a(i,j)=a(i-1,j)+a(i,j-1)
 200   CONTINUE
 100 CONTINUE
```

図 11 ループ傾斜適用前のソースコード
Fig. 11 Source code without the loop skewing.

```
DO 100 i = 2, 6
       DO 200 j = i+2, i+6
         a(i,j-i)=a(i-1,j-i)+a(i,j-1-i)
 200   CONTINUE
 100 CONTINUE
```

図 12 ループ傾斜適用後のソースコード
Fig. 12 Source code with the loop skewing.

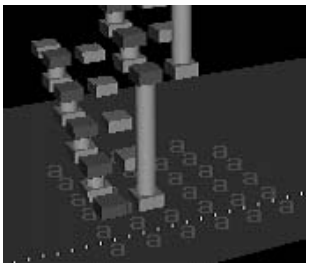

図 13 図 11 の **DDV**
Fig. 13 **DDV** of Fig. 11.

いループが混在する場合，ループ交換可能である．

ループ交換は，実際には並列度の粒度調節やデータ依存解消を行うための前処理や後処理等で使われるが，ここでは最も単純なパターンをあげている．たとえば並列度の粒度調節の場合は，ループ繰越しのポールを持たないループのみが多重ループにある場合に適用される．

4.3 ループ傾斜

ループ傾斜とは，二重ループ☆の外側ループのインデックス変数を内側ループのインデックス変数定義域に加え，同時に配列参照の整合性を保たせることで得られるユニモジュラ変換[6] の一種である．**図 11** と **図 12** にループ傾斜の例を示す．

図 13 は図 11 の配列参照を可視化したものである．このように，赤いキューブ（ライト）が複数の青いキューブ（リード）に囲まれていて，複数の長さのポールが存在するとき，波頭計算[8] となる．波頭計算はループ傾斜を適用することにより並列化が可能となる．適用後の図 12 では，内側のループが DOALL 型

☆ 多重ループの場合，ループ交換により隣りあう二重ループとして考えることができる．

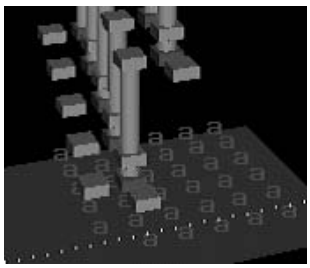

図 14 依存ベクトル $(1, 1)$ の場合の **DDV**
Fig. 14 **DDV** with the dependence vector $(1, 1)$.

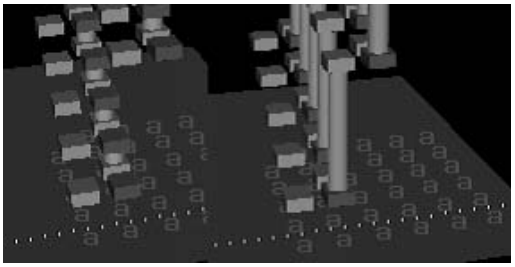

図 15 依存ベクトル $(-1, 1)$ の場合の **DDV**
Fig. 15 **DDV** with independence vector $(-1, 1)$.

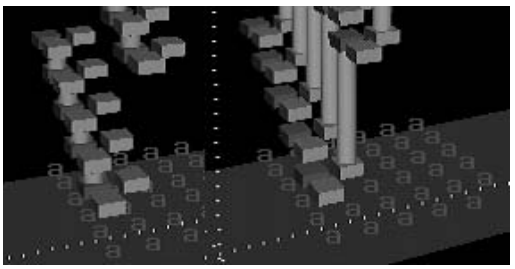

図 16 依存ベクトル $(1, -1)$ の場合の **DDV**
Fig. 16 **DDV** with independence vector $(1, -1)$.

並列実行が可能である．

- **ループ傾斜パターン：**二重ループの **DDV** において赤いキューブが複数の青いキューブに囲まれており，内側ループ外側ループともにループ繰越しのポールを持つ場合，ループ傾斜適用可能である．

図 11 は配列 a に対する依存ベクトル[3] が $(-1, -1)$ の場合のパターンである．**図 14**，**図 15**，**図 16** はそれぞれ依存ベクトルが $(1, 1)$，$(-1, 1)$，$(1, -1)$ であるときのパターンである．いずれもループ傾斜適用可能である．

波頭計算は初学者には並列化不可能と思われがちであるが，本パターンを利用することで並列化を容易に行うことができる．

4.4 ループ細分

図 17 と **図 18** はループ細分の例である．**図 19** は図 17 の配列参照の可視化を示すが，ループ繰越しの

```
DO 100 i = 1, 9
        a(i+2) = a(i) + 1
 100 CONTINUE
```

図 17 ループ細分前のソースコード
Fig. 17 Source code without the strip mining.

```
DO 100 si = 1, 8, 2
       DO 200 i = si, si+1
         a(i+2) = a(i) + 1
 200   CONTINUE
 100 CONTINUE
     DO 300 i = 9, 9
       a(i+2) = a(i)+1
 300 CONTINUE
```

図 18 ループ細分後のソースコード
Fig. 18 Source code with the strip mining.

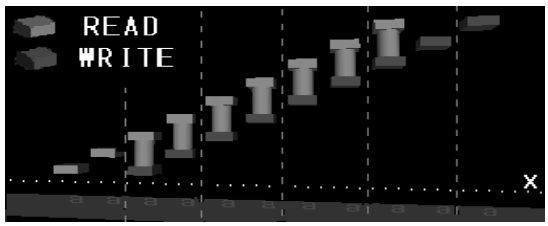

図 19 図 17 の **DDV**
Fig. 19 **DDV** of Fig. 17.

短いポールが多数あり，並列化できない．図 18 では，二重ループの内側のループが DOALL 型並列化可能であることが分かる．このように，ある（単一）ループ中に同じ長さのポールが多数ある場合，その長さでループ細分を適用することで並列化が可能となる．

- **ループ細分パターン：**単一ループの **DDV** で，特定の配列に関する同じ長さのループ繰越しのポールが多数連続してある場合，ループ細分を適用可能である．

ループ細分は任意のループに適用可能である．ここではイタレーション長をデータ依存距離の長さに合わせてデータ依存を解消した．この手法は正確には巡回縮小[9] をループ細分することによって実現したものである．実際のループ細分は，スケジューリングの粒度を調節したり，データ通信のタイミングを合わせるのに用いられるが，NaraView を使った初学者への利用例としてあげたものである．

4.5 ループ皮むき

ループ皮むきはループの最初もしくは最後の特定のデータ依存を残りのイタレーションから取り去る変換手法である．**図 20** の配列参照を可視化した結果が **図 22** 上である．配列要素 $a(1)$ 上にイタレーション全体にわたるライフタイム・ポールが観測される．つまり，配列 a の残りの全要素が $a(1)$ とフロー依存関係を形成し，全体の並列化を阻止している．ループ皮むきを行い，**図 21** のように変換することにより，図 22 下に示されるようにポールが除去され，DOALL 型並列化が可能となる．このように特定の配列要素が全イタレーションにわたってデータ依存を引き起こす場合，ループ皮むきが有効である．

```
DO 100 i = 1, 10
        a(i) = a(i) + a(1)
 100 CONTINUE
```

図 20 ループ皮むき適用前のソースコード
Fig. 20 Source code without the loop peeling.

```
a(1) = a(1) + a(1)
     DO 100 i = 2, 10
        a(i) = a(i) + a(1)
 100 CONTINUE
```

図 21 ループ皮むき適用後のソースコード
Fig. 21 Source code with the loop peeling.

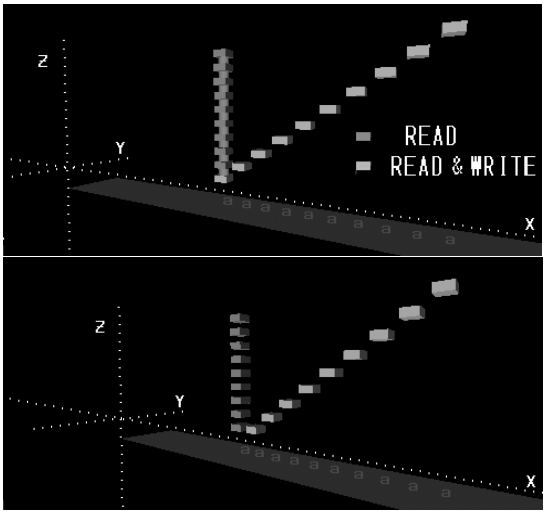

図 22 ループ皮むき適用前後の **DDV**
Fig. 22 **DDV** with and without the loop peeling.

- **ループ皮むきパターン：**あるループの **DDV** で特定の配列要素の上にイタレーション全体にわたるポールが存在し，他にループ繰越依存がない場合，ループ皮むきを適用して並列化が可能となる．

イタレーション全体にわたるループ繰越依存のポールが，イタレーションの最初や最後ではなく，途中にある場合，ループ細分やループ分配を利用してからループ皮むきを適用する．

5. 関連研究

ParaScope editor[10] は Rice 大学で開発された有名な対話型コンパイラシステムである．これは以下のような特徴を持つ．

- データ依存関係をテキスト表示で見ることができる．
- ユーザは表示されたデータ依存関係を選んで，その依存関係が実際にはないとシステムに知らせることができる．
- ユーザはプログラムの中からループを選び，そのループに対して適用する並列化手法をメニューから選ぶことができる．

実際にこのシステムを使ってみたユーザは，非常に良いシステムだがさらなる改良ができそうだという感想を持っている．ParaScope が提供するインタラクションはすべてテキストベースであり，グラフィカルな表示を望むユーザの声があったことが[10] で報告されている．

SUIF[11] はスタンフォード大学で開発された自動並列化コンパイラで，SUIF explorer[12] はその対話型並列化ツールである．このツールでは，以下の可視化が行われている．

- 関数呼び出しグラフ（function call graph）
- ソースコードの図的表現．ソースコードの 1 行をその長さに応じた直線にして何千行ものソースコードを一目で概観できるようにしている

また，ソースコードをテキスト表示したとき，色分けすることによってユーザが重要な部分を見分けられやすいようにしている．並列化支援としてはプログラムスライシングの技術を使ってユーザに必要な情報を提供している．

Ishihara ら[13] は OpenMP に対応した対話型並列化支援システムを提案している．これは Emacs を使ってプログラムのソースとデータ依存関係の情報を表示し，それをもとにユーザがソースプログラムを書き換えたりあるいは OpenMP のディレクティブを書き込んだりするというものである．

Calidonna らの GPE（Graphical Parallelizing Environment）[14] は Parafrase-2[16] を使い，HTG[17] のグラフ表示（HTGvis）とソースコードを関連付けたものを使って並列化を支援する．OpenMP に対応していて OpenMP のディレクティブを挿入するための支援もできる．

こうしてみると，並列化支援のための対話型のコンパイル環境の研究は最近あまり進んでいない．近年では OpenMP 対応にする研究が行われているが，どれも本質的な部分はソースコードとデータ依存関係の情報をテキストで表示し，ユーザに指示をあおぐというものである．プログラムの依存関係のグラフ表示を行うという研究は 1990 年代にいくつかされているが（たとえば文献 15)），それからあまり活用されていないようである．

本研究では，並列化の対話的処理という目標は同じものであるが，いかにして並列プログラミング初学者に難解なデータ依存解析やループ変換理論を強要せず直観的に理解しやすい支援を行えるか，という観点で開発を行っており，他の関連研究とは一線を画する．

6. 結　論

並列化支援可視化システム NaraView を使って，ループ中のデータ依存を可視化し，対応する並列化手法に従って可視化パターンの分類を行った．これにより，並列プログラミングの初学者が対話的に目視で適切な並列化を行える環境が可能であることを示された．

今後の課題はより詳細かつ複合的なデータ依存関係を分析し分類することで，NaraView を並列プログラミングを対話的に行うためのシステムとして再構築することである．また，このような対話的支援システムでは，インタラクションに対する評価が不可欠である．つまり，ユーザが何らかの並列化手法を選択した場合，それがどの程度有効であるかを何らかの指標をもとに通知すべきである．これは性能指向型対話的並列プログラミング支援環境として実現されるべきであり，将来の大きな研究課題の 1 つである．

謝辞 プログラム例とその可視化データを作成してくれた岩坂麻美さんに感謝いたします．

参考文献

1) Haneda, M., Sasakura, M., Nagashima, U., Kunieda, Y. and Joe, K.: Collaboration of Parafrase-2 and NaraView for Effective Parallelization Supports, *PDPTA2000*, pp.635–641 (2000).
2) Sasakura, M., Joe, M., Kunieda, Y. and Araki, K.: NaraView: An Interactive 3D Visualization System for Parallelization of Programs, *International Journal of Parallel Programming*, Vol.27, No.2, pp.111–129 (1999).
3) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).
4) 笹倉万里子，木和田智子，城　和貴，荒木啓二郎：並列化支援視覚化システム NaraView に

おけるプログラム情報の 3 次元表示法，情報処理学会並列処理シンポジウム JSPP96，pp.267–274 (1996).
5) 山口智美，石内寿子，岩坂麻美，羽田昌代，庄野逸，城　和貴：自動並列化コンパイラ PROMIS-NWU の概要，情報処理学会計算機アーキテクチャ研究会，ARC-144-14, pp.79–84 (2001).
6) Banerjee, U.: *Loop Transformations for Restructuring Compilers: the foundations*, Kluwer Academic Publishers (1993).
7) Bacon, D.F., Graham, S.L. and Sharp, O.J.: Compiler Transformations for High-Performance Computing, *ACM Computing Surveys*, Vol.26, No.4, pp.345–420 (1994).
8) Wolf, M.E. and Lam, M.S.: A Loop Transformation Theory and an Algorithm to Maximize Parallelism, *IEEE Trans. Parallel & Distributed Syst.*, Vol.2, No.4, pp.452–471 (1991).
9) Polychronopoulos, C.: Parallel Programming and Compilers, Kluewer Academic Press (1988).
10) Hall, M.W., Harvey, T.J., Kennedy, K., McIntosh, N., McKinley, K.S., Oldham, J.D., Paleczny, M.H. and Roth, G.: Experiences using the ParaScope Editor: An interactive parallel programming tool, *SIGPLAN Notice*, Vol.28, No.7, pp.33–43 (1993).
11) Wilson, R.P., French, R.S., Wilson, C.S., Amarasinghe, S.P., Anderson, J.M., Tjiang, S.W.K., Liao, S.-W., Tseng, C.-W., Hall, M.W., Lam, M.S. and Hennesy, J.L.: SUIF: An infrastructure fro research on parallelizing and optimizing compilers, *ACM SIGPLAN Notices*, Vol.29, No.12, pp.31–37 (1994).
12) Liao, S.-W. Diwan, A., Bosch Jr., R.P., Ghuloum, A. and Lam, M.S.: SUIF Explorer: An interactive and interprocedural parallelizer, *7th ACM SIGPLAN*, pp.37–48 (1999).
13) Ishihara, M., Honda, H., Yuba, T. and Sato, M.: Interactive parallelizing assistance tool for OpenMP: iPat/OMP, *5th European Workshop on OpenMP EWOMP2003*, pp.21–29 (2003).
14) Calidonna, C.R., Giordano, M. and Furnari, M.M.: A graphic parallelizing environment for user-compiler interaction, *13th ACM ICS*, pp.238–245 (1999).
15) Dow, C.-R., Chang, S.-K. and Soffa, M.I.: A visualization system for parallelizing programs, *IEEE SuperComputing 92*, pp.194–203 (1992).
16) Polychronopoulos, C.D., Girkar, M., Haghighat, M.R., Lee, C.L., Leung, B. and Schouten, D.: Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors, *ICPP*, Vol.2, pp.39–48 (1989).
17) Girkar, M. and Polychronopoulos, C.D.: The Hierarchical Task Graph as a Universal Intermediate Representation, *International Journal of Parallel Programming*, Vol.22, No.5, pp.519–551 (1994).

(平成 17 年 8 月 28 日受付)
(平成 17 年 12 月 26 日採録)

山口　智美（学生会員）

奈良女子大学理学部情報科学科卒業，2003 年同大学大学院人間文化研究科情報科学専攻博士前期課程修了，現在同研究科複合現象科学専攻複合情報科学講座博士後期課程在学中．

笹倉万里子（正会員）

京都大学工学部情報工学科卒業，(財) 京都産業情報センター，(財) 京都高度技術研究所勤務を経て，1995 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．1996 年同大学院大学情報科学研究科博士後期課程中退．同年岡山大学工学部情報工学科助手．現在に至る．博士（工学）．並列化支援のための視覚化システムに関する研究に従事．日本ソフトウェア科学会，人工知能学会，IEEE-CS 各会員．

城　　和貴（正会員）

大阪大学理学部数学科卒業．日本 DEC，ATR 視聴覚研究所（日本 DEC より出向），(株) クボタ・コンピュータ事業推進室で勤務の後，1993 年奈良先端科学技術大学院大学情報科学研究科博士前期課程入学，1996 年同研究科後期課程修了，同年同研究科助手．1997 年和歌山大学システム工学部講師，1998 年同助教授．1999 年奈良女子大学理学部情報科学科教授，現在に至る，工学博士．情報処理学会論文誌「数理モデル化と応用」編集委員長．