ループ内依存関係の視覚的パターン化の試み

岩坂 麻実 † 山口 智美 ‡ 庄野 逸 ‡ 笹倉 万里子 ※ 城 和貴 †

† 奈良女子大学理学部情報科学科
‡ 奈良女子大学大学院人間文化研究科
※ 岡山大学工学部情報科学科

並列計算機の普及に伴い逐次プログラムを並列プログラムに書き換える必要があるが，現在のところ，プログラマに対して高度な技術や経験が要求される．本研究では，並列支援視覚化システム NaraView を用いて，与えられたプログラムのデータ依存関係を視覚的パターン化し，適切な変換手法に対応づけることを考案する．本稿では主に，並列化コンパイラで一般に使用されているループ変換手法について取り上げ，具体的に視覚化されたプログラム中のループのデータ依存状態から有効なループ変換手法を検証する．

Attempt to Classify Visualized Data Dependence

Asami Iwasaka † Tomomi Yamaguchi ‡ Hayaru Shouno ‡ Mariko Sasakura * Kazuki Joe †

†Department of Information and Computer Siences Nara Women's University
‡Graduate school of Human Culture Nara Women's University
* Department of Information Technology Okayama University

Recently, in the field of scientific calculation, the tendency to introduce parallel computer becomes increase. Therefore, converting methods from conventional sequential programs to correspondent parallel ones are required. However, vast amounts of skill and knowledge for the parallel programming are needed to programmer. In this study, we propose introducing the 3D visualization system called NaraView as those programming environment. NaraView analyze the data dependence in the programming code, and visualize their relations. Since the parallelizable code have typical features, NaraView would reveal them visually. We showed the effective parallelizing method, which have those typical features in the loop structure with concrete program code.

1 はじめに

近年，並列計算機の導入が急激に増加しており，既存のアプリケーションを並列計算機で使用するためには，逐次プログラムを並列プログラムに書き換える必要がある．しかしながら，各プログラムに対して構造を把握し再構成を行うことは容易ではない．現在のところ，高度な技術や経験を持つプログラマが時間をかけて大規模なプログラムに対してそれぞれ有効な並列化を考案し実装しなければならない．NaraView[2] はプログラムを視覚化することによって，ユーザーにその構造を迅速かつ正確に伝える並列化支援視覚化システムである．本研究では，主として並列プログラムを目的としたループ変換手法によるデータ依存関係の相違を視覚化し，プログラム構造や依存関係と有効な変換手法の関連付けを行うことを目的とする．これらの情報によりプログラマは視覚化されたプログラムのループの構造から適切なループ変換手法を容易に選択することが可能になり，経験の少ないプログラマでも逐次プログラムから並列プログラムへの変換が可能になると考えられる．

2 データ依存関係とループ変換手法の対応

NaraView では，プログラム情報を 3 つのビュー：Program Structure View(PSV), Source Code

View(SCV), Data Dependence View(DDV) で表示することが可能である．DDV では，変数や配列要素をキューブで，その依存関係をポールで表示し，その色によってアクセス方法，依存の種類を識別する．ユーザーはこれらのビューを併用することによって有効な並列化を検討する．ここでは並列化コンパイラで一般に使用されるループ変換手法，Loop distribution, Loop interchange, Loop skewing, Strip mining, Loop peeling を取り上げる [1]．プログラム中の変数や配列要素について依存関係を調べ，その依存関係を保持したまま並列化を行うことは極めて難しい．そこで，各ループ変換手法についてデータ依存関係の視覚化を行い，依存関係の形状から正当かつ最適なループ変換を検討する．

2.1 Loop distribution

図 1 では，x(i) と x(i-1) 間にクロスイタレーション依存が存在する．そこで，Loop distribution を適用することにより，図 2 のように 2 つにループを分割する．その結果，クロスイタレーション依存のない S1 については並列実行が可能になる．

図 3 の X についての紫キューブ（read&write アクセス）から青キューブ（read アクセス）の間にある緑のポールはクロスイタレーション依存が全体の並列実行を阻害している事を示している．このようにひとつの配列要素にクロスイタレーションが存在する場合には，それを分離するために Loop distribution を適用することが有効である．

```
      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 1. Sorce code (Before Loop distribution)

```
      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 2. Sorce code (After Loop distribution)

図 3. Flow Dependence
(Before Loop distribution)

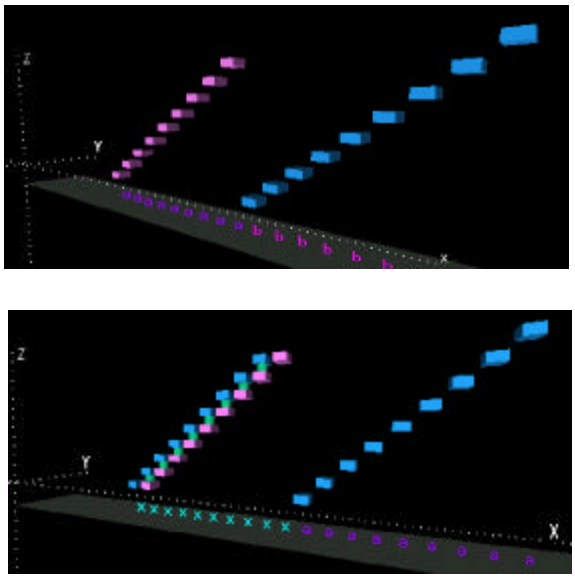

図 4. Flow Dependence
(After Loop distribution)

2.2 Loop interchange

Loop interchange の例を図 5，図 6 に示す．

図 11 左図に見られるように配列 b について逆依存が存在するが，X 軸方向すなわち j のループについて並列化が可能である．j のループは内側であり，並列化の効果は外側のループに適用した場合に高くなるので，Loop interchange を行う．変換後の逆依存関係は図 11 右図のようになり，Y 軸方向すなわち外側の j のループに関して並列化が可能になることがわかる．

このように，一方向にのみ並列化が可能でありそれが内側のループである場合，Loop interchange が有効である．

```
      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 5. Source code (Before Loop interchange)

```
      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 6. Source code (After Loop interchange)

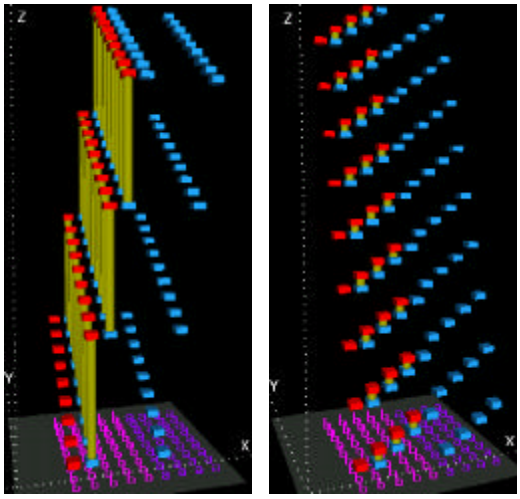

図 7. Anti Dependence
(左) Before Loop interchange
(右) After Loop interchange

2.3 Loop skewing

Loop skewing の具体例を図 8，図 9 に示す．図 10 のように同一イタレーション内で write 文の赤キューブが 2 方向以上の read 文の青キューブで囲まれており，その赤キューブの上に青キューブが存在しそれらの間にフロー依存がある場合，wave front 計算があることを示しており，Loop skewing が適用可能である．

```
      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 8. Source code (Before 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 9. Source code (After Loop skewing)

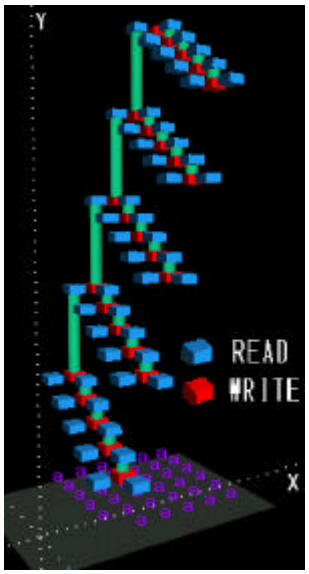

図 10. Flow Dependence

2.4 Strip mining

長さ 2 での Strip mimnig の具体例を図 11，図 12 に示す．図 13 では X-Z 平面にのみキューブが存在し，これは一次元ループであることがわかる．また，write 文の赤キューブの上に他のキューブがひとつ存在した後，元のキューブの上に read 文である青キューブが現れている．これは，ある配列要素について依存距離 2 のフロー依存が存在することを表しており，長さ 2 で Strip mining を適用することができる．赤キューブと青キューブの間にさらに多くの別のキューブが存在する場合はそのキューブ数に 1 を足した数で Strip minig が適用可能である．

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

図 11. Source code (Before Strip mining)

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

図 12. Source code (After Strip mining)

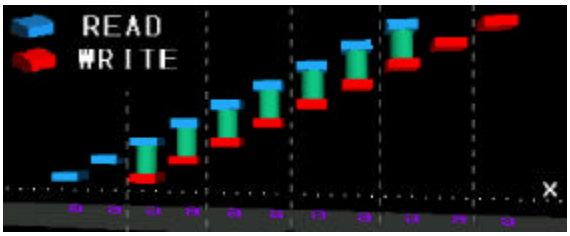

図 13. Flow Dependence

2.5 Loop peeling

図 14 のデータ依存を視覚化すると図 16 上図のようになる．紫キューブは a の配列要素を表しており，左端のキューブは a(1) である．このループでは z 軸方向に上昇，すなわちループの進行とともにアクセスされるキューブは進むが，a(1) に対してはすべてのキューブが緑のポールでつながれており，フロー依存が存在することがわかる．これは，a(1) に対する依存のためにループの並列化が阻害されていることを意味する．よって，このループに Loop peeling を適用すると図 16 下図にみられるように a(1) へのフロー依存がなくなるため，並列実行が可能になる事がわかる．

このようにあるひとつの紫キューブの上に多数の青キューブが存在し，それらがポールでつながれている場合，Loop peeling を適用する事が望ましいと考えられる．

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

図 14. Source code (Before Loop peeling)

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

図 15. Source code (After Loop peeling)

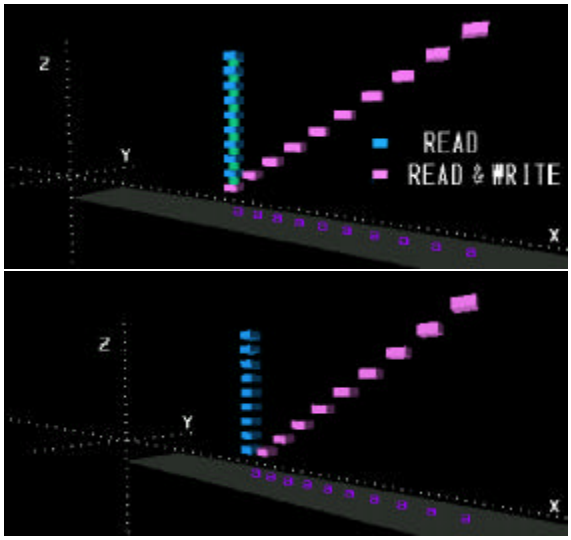

図 16. Flow Dependence
(上)Before Loop peeling
(下)After Loop peeling

3 結論

並列化支援視覚化システム NaraView を用いてプログラム中のループの依存関係を視覚化を行い，有効なループ変換手法との関連付けを行った．ユーザーはこれらの視覚化された依存情報をもとに各プログラムに対して適切な並列化を見出すことができ，逐次プログラムから並列プログラムへの変換が迅速かつ容易になると考えられる．

今後の課題として，より詳細な依存関係の解析および分類，選択した並列化手法の実装などの NaraView の機能拡張，などが挙げられる．

参考文献

[1] 中田　育男, ”コンパイラの構成と最適化 ”, 朝倉書店,1999

[2] M.Sasakura, K.Joe, Y.Kunieda, and K.Araki, NaraView: An interactive 3D visualization system for parallelization of programs, International Journal of Parallel Programming, 27(2): 111-129,1999