計算機アーキテクチャ 134－25
（１９９９．８．４）

配列参照パターンによるプログラム並列化・最適化支援

奥田 宗継 * 中西 恒夫 * 笹倉 万里子 ‡ 城 和貴 † 福田 晃 *

narafrase@is.aist-nara.ac.jp

* 奈良先端科学技術大学院大学情報科学研究科
‡ 岡山大学工学部情報工学科
† 奈良女子大学理学部情報科学科

概要

「控え目な並列化・最適化」を行う自動並列化コンパイラは，しばしばプログラムの並列化・最適化に失敗し，出力されたプログラムに並列化・最適化の余地が残っていることが多い．プログラマが並列化・最適化に必要な情報を取得し，並列化・最適化を手動で適用するインタラクティブコンパイル環境の実現が望まれる．本稿では，並列化支援視覚化システム NaraView を用いてループにおける配列参照パターンを視覚化し，並列化・最適化を施す方法に関する議論を行う．

Program Parallelization/Optimization Support with Array Reference Patterns

Noritsugu Okuda * Tsuneo Nakanishi * Mariko Sasakura ‡ Kazuki Joe † Akira Fukuda *

* Graduate School of Infomation Science , Nara Institute of Science and Technology
‡ Faculty of Engineering , Okayama University
† Faculty of Science , Nara Women's University

Abstract

Parallelizing compilers often fail in program parallelization and optimization due to their conservative behaviors, thus not a little part of a given program can still remain to be parallelized or optimized. Manual parallelization and optimization with interactive compilation environments will be one of reasonable solutions to that problem. In this paper we visualize array reference patterns in loops with NaraView Interactive Program Visualization System and present a scheme to parallelize and optimize the program interactively using its visualized images.

1 はじめに

自動並列化コンパイラは，入力された逐次プログラムを自動的に並列化されたプログラムに書き換えるコンパイラである．自動並列化コンパイラを用いることにより，プログラマは複雑な並列プログラミングをする必要がなくなり，これまでのように逐次プログラミングを行うのみで並列プログラムを作成することができる．また過去に開発された膨大な量の逐次プログラムを修正することなく並列実行させることができる．しかしながら，しばしば自動並列化コンパイラはプログラムの並列化・最適化に失敗し，出力されたプログラムには並列化・最適化を行う余地が残ってい場合が多い．プログラムが十分に自動的に並列化・最適化されないひとつの原因として,「控え目の並列化・最適化」[1] を挙げることができる．すなわち，プログラムにある変更を加える際に，変更後のプログラムが正しく実行されると断定できない限りは変更を加えないというものである．

例えば，自動並列化コンパイラでは依存解析 [2] が行われる．依存関係とは，あるプログラムが並列に実行できるかどうかを判定するための主要な情報の1つである．依存解析を行うことにより，プログラムが正しい結果を出力するためにはプログラムのどのタスクをどのタスクより先に実行しなければならないかというタスク間の依存関係が得られる．依存関係にあるタスクは並列に実行できない．依存解析を「控え目」に行うとは，タスク間に依存関係がないと断定できない限りは依存があるものとみなすということである．依存解析を控え目に行うことにより並列化・最適化が妨げられる．

以上の背景に立ち，我々は，プログラマにプログラムを分かりやすく視覚的に提示して，プログラマからコンパイルに関する指示やヒントの提供を受け，より並列化・最適化を行うインタラクティブコンパイル環境 NaraView[3] を構築している．

科学技術計算プログラム中のループには多くの並列性が存在する．また，ループ中では配列変数がよく参照されるが，配列変数に関する依存解析は難しい問題であり，控え目の解析のためにしばしば自動並列化コンパイラはループの並列化・最適化に失敗する．そこで本稿では，配列変数の参照を視覚化し，プログラマが並列化・最適化に関するヒントを獲得し，並列化・最適化をインタラクティブに施す方法について議論する．

以後，第2節では NaraView の概要について述べる．第3節では，NaraView のデータ依存関係ビューの概要と配列の参照情報を得る手法の1つを説明する．第4節では NaraView のデータ依存関係ビューを用いたループ変換手法と最適化に関する枠組を述べる．第5節で結論と今後の課題を述べる．

2 NaraView の概要

我々はソースプログラムを入力として，プログラマがプログラムを手動で並列化・最適化する（チューニング）際に必要な情報を視覚化するシステム NaraView を開発している．NaraView は，プログラムのどこが並列化され，どこが並列化されていないかを視覚的に表示し，並列化が行われていない部分があれば，表示されたプログラム情報から NaraView を媒体としてインタラクティブにプログラムを並列化・最適化するのに用いられる． NaraView はプログラム構造，データ依存，制御依存等の情報を3次元的に視覚化する．NaraView がプログラム情報を3次元表示するのは，大規模なプログラムの情報を限られた表示面上に視覚化するためである．現在 NaraView には，プログラム構造ビュー，データ依存関係ビュー，ソースコードビューの3つのビューが実装されている．

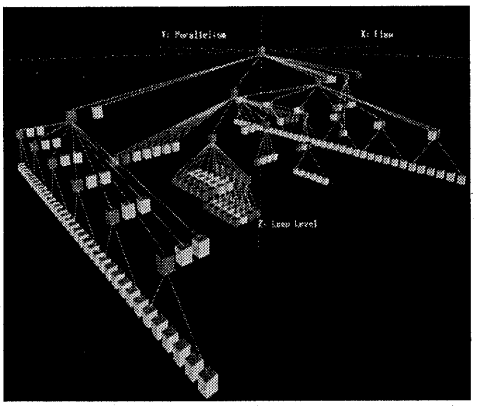

図 1: プログラム構造ビュー

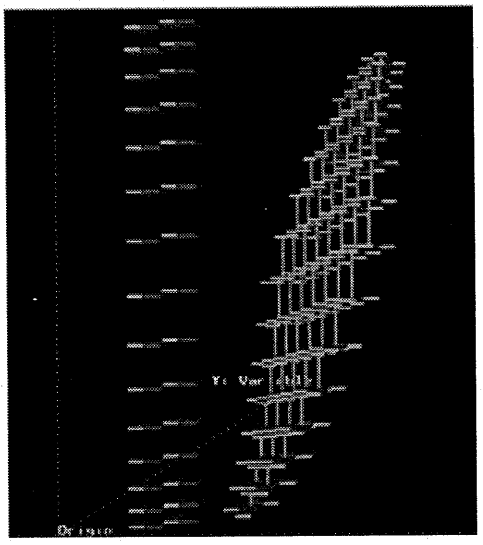

図 2: データ依存関係ビュー

- プログラム構造ビュー（図 1）：視覚化するプログラムの時間的な流れ，並列性，ループネスティングレベルを一目で理解することができるよう表示する．3 次元それぞれの座標は以下のような情報を表す．
  - X ：プログラムの流れ
  - Y ：並列度
  - Z ：ループネスティングレベル
- データ依存関係ビュー（図 2）：配列や変数の書き込み，読み込みによって発生する依存情報を 3 次元的に表示する．プログラム構造ビューから呼び出されるループ中の配列への参照を色のついた立方体，これらの参照によって生じる依存を円柱で表示する．
  - X , Y ：配列や変数の配置
  - Z ：イタレーション
- ソースコードビュー：プログラム構造ビューから呼び出されるプログラムのソースコードを表示する．

プログラマは，プログラム構造ビューを見てプログラム全体の並列実行状態を直感的に把握することができる．例えば，(自動並列化コンパイラ等によって) すでに並列化・最適化されたプログラムであればどのように並列化・最適化 されているかをプログラム構造ビューによって理解し，次にどの部分を並列化・最適化すればよいかを探ることができる．そして，未並列化部分があれば対応するループのノード（各ノードは，プログラムの 1 文に相当する）をクリックしてソースコードビューを呼びだし，プログラムの詳細を追うことができる．さらに同じようにループのノードをクリックしてデータ依存関係ビューを呼び出し，どのループ内のどの変数（配列要素）への参照が原因となってどのイタレーション間に依存関係があるのかを追跡することができる．プログラマは，ソースコードビューとデータ依存関係ビューからどのように並列化・最適化を行うかをコンパイラに指示する．

本稿では，上述のデータ依存関係ビューにより視覚化された配列参照パターンから，プログラマが並列化・最適化に関するヒントを獲得し，並列化・最適化をインタラクティブに施す方法について議論する．

3 配列参照パターンの視覚化

3.1 データ依存関係ビュー

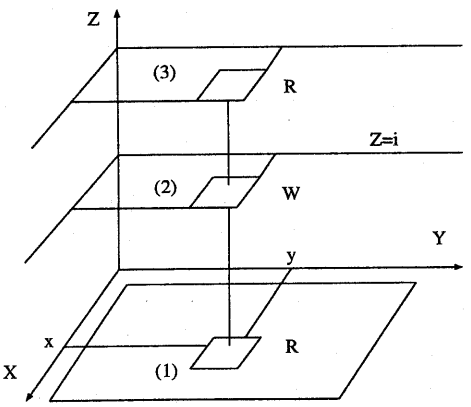

図 3: データ依存関係ビューの概念図

データ依存関係ビューは個々のループネストにおける変数（配列要素）への参照を色のついた立方体として，これらの参照によって生じるイタレーショ

ン間でのデータ依存を円柱として視覚化する．

図3はデータ依存関係ビューの概念図を表したものである．まず配列は $X-Y$ 平面上に配置され，Z 座標にはループネストのイタレーションが対応づけられる．$Z=i$ の $X-Y$ 平面は i 番目に実行されるイタレーションに対応する．点 (x, y) に対応する配列要素が i 番目にイタレーションのおいて書き込みが行われる場合，(x, y, i) には赤色の立方体が配置される．同様に読み込みが行われる場合，(x, y, i) には青色の立方体が配置される．また，同一のイタレーション内で書き込みと読み込みが同時に行われる場合，(x, y, i) には紫色の立方体が配置される．また，配列の依存関係を表す円柱には以下ようなものがある．

- ライフタイムポール：同一の配列要素において書き込みの立方体から次の書き込みの直前の読み込みの立方体をつなぐ円柱
- アンチディペンデンスポール：同一の配列要素において，読み込みの立方体からそれ以後の最初の書き込みの立方体をつないだもの

ライフタイムポールは配列変数の値の寿命を，アンチディペンデンスポールは逆依存の存在を表している．

また，データ依存関係ビューは以下の2種類の Z 軸に垂直な半透明の面を表示できる．

- イタレーショングリッド：ネスト内の各ループの最初のイタレーションを表す．
- AVD（Array-Variable Dirpositoin）マップ：配列の $X-Y$ 平面上への配列変数の配置を表示．

3.2 配列参照パターンの取得

配列参照パターンを取得する方法としては，コンパイル時に配列参照に関する情報を得る方法とプログラムの実行時に情報を得る方法の2つが考えられる．コンパイル時に情報を得る場合はプログラムにある記述から配列参照に関する情報を得なければならない．つまり，実行しなければ得ることができない情報（実行時にプログラマが入力する場合等）がループの境界値や配列の添字に参照されていると，コンパイル時にはその配列に関して参照情報を得ることができない．一方，プログラムの実行時に配列参照に関する情報を得る場合は，逆に前述のような情報の不足が起ることがない．そのため対象とするプログラムを選ばない点で有利である．但し，プログラムを実行しなければならない分，時間を要する．

本稿では実行時に配列参照パターンを取得することとし，得られた配列参照パターンをNaraViewで3次元的に視覚化する．自動並列化コンパイラは，コンパイル時に取得できる情報を用いて並列化・最適化を行うためにコンパイル時に不明な情報をループ境界値や配列の添字が参照している場合，しばしば並列化・最適化に失敗してしまう．そこで本稿のように実行時の配列参照パターンを用いることで，より並列化・最適化を施すことができると考えられる．

以下，実行時に配列の参照情報を取得する方法について説明する．例えば図4のようなプログラムがあった場合，図5のように配列参照のログを出力するコードを埋め込む．

```
for(i = 0; i = 3; i++){
    for(j = 0; j = 3; j++){
        a[i][j] = ··· ;
    }
}
```

図4: サンプルプログラム1

ログの書式を以下に示す．

name :*access_type* :*index*1 ;*index*2 :*iteration*

但し，

- *name* ：配列変数名
- *access_type* ：配列参照種別（読み込まれた場合'R'，書き込まれ た場合'W'）
- *index* ：配列の添字の値
- *iteration* ：イタレーションの番号

ログ出力コードを埋め込んだプログラムを実行することにより，そのプログラムの配列参照のログが得られる．得られたログを NaraView で読み込み可能にするように変換する．

```
for(i = 0; i = 3; i++){
    for(j = 0; j = 3; j++){
        a[i][j] = ··· ;
        /* LOGGING BEGIN */
        printf("a:W:%d;%d:%d\n",
                                i,j,ixxx);
        ixxx++;
        /* LOGGING END */
    }
}
```

図 5: サンプルプログラム 2

4 データ依存関係ビューによる並列化・最適化支援

4.1 データ依存関係ビューの解釈

データ依存関係ビューを Z 軸に垂直な平面 S で切断するものと仮定する．このとき S は 0 本以上のライフタイムポールを切断する．ライフタイムポールは，変数の値を定義するイタレーションに対応する Z 軸に垂直な平面から，その値を使用するイタレーションに対応する Z 軸に垂直な平面の間に設けられるものである．すなわちライフタイムポールの高さは当該変数の参照に伴うフロー依存の依存距離にほかならない．(但し，イタレーションを表す平面は常に格子面上に間隔 1 で順番に配置されているものとする．) また，ライフタイムポールの S による断面積は，S より低い平面に対応するイタレーションから，S より高い平面に対応するイタレーションへのデータの転送量に比例するものと考えることができる．これらの観察は，後述するように，データ転送の最適化やレジスタ割当，キャッシュ利用の効率化等に役立てることができる．

4.2 ループ変形手法の適用

ループ変形手法の適用によりデータ依存関係ビューの表示は更新されることになる．

Loop Interchange [4], Loop Reversal [4], Loop Tiling [4] 等のループのイタレーションの実行順序を変更するループ変形手法を適用する場合，データ依存関係ビューのイタレーション平面が並べ替えられることとなる．Loop Unrolling [4] を適用する場合は隣接する複数のイタレーション平面が合体され，逆に Loop Rerolling [4] を適用する場合は 1 枚のイタレーション平面が隣接する複数のイタレーション平面に分割されることとなる．

また複数のループをひとつのデータ依存関係ビュー上に表現する場合，Loop Distribution [4] や Loop Fusion [4] を考えることができる．Loop Distribution を適用する場合はイタレーション平面が分割され，Loop Fusion を適用する場合はイタレーション平面が合体される．但し，それぞれ Loop Unrolling と Loop Rerolling の場合とは異なり，合体・分割されるイタレーション平面は隣接するものとはならない．

以上の考察から，イタレーション平面の並べ替え，移動，合体，分割の操作がデータ依存関係ビュー上でできるようになれば，それらの操作に対応する様々のループ変形手法をデータ依存関係ビュー上で適用することが可能となる．

4.3 データ依存関係ビュー上での最適化

前々小節で考察した通り，ライフタイムポールの Z 軸に垂直な平面 S による断面積は，S より低い平面に対応するイタレーションから，S より高い平面に対応するイタレーションへのデータの転送量に比例するものと考えることができる．

ゆえに以下の操作によりプロセッサ間通信の最適化を検討することができる．

1. プロセッサ P_1 に割り当てられるイタレーション平面ならびにプロセッサ P_2 に割り当てられるイタレーション平面のみを残す．

2. P_1 に割り当てられるイタレーション平面が S より下，P_2 に割り当てられるイタレーション平面が S より上になるようにイタレーション平面を並べ替える．

3. ライフタイムポールの S による断面積が最小になるようにイタレーション平面の並べ替え，移動，合体，分割を施す．

また，プロセッサへのイタレーションの割当が決まれば，以下の操作によりプロセッサ内でのレジスタ割当やキャッシュ利用の効率化を図ることができる．

1. プロセッサ P に割り当てられるイタレーション平面のみを残す．

2. 各イタレーション平面の間に S を置いたとき，ライフタイムポールの S による断面積が最小になるようにイタレーション平面の並べ替え，移動，合体，分割を施す．

前者の操作では S は空間的分割を，後者の操作では S は時間的分割を意味していることに注意されたい．

5 おわりに

本稿ではプログラムの並列化支援視覚化システム NaraView のデータ依存関係ビューを用いて，プログラム中の配列参照情報を視覚化し，プログラムの並列化・最適化をプログラマ自身によって行う枠組を示した．

今後の課題として本稿第 4 節で議論したようなプログラマによる最適化を効率的に行えるよう，データ依存関係ビューを改良していきたい．実アプリケーションにおいてはしばしば大規模な配列が扱われる．しかし，アプリケーションの配列参照パターンを視覚化する負荷は極めて大きい．大規模な配列の参照パターンの視覚化法は今後の研究課題である．また，さらに多くの検証を重ね，配列参照情報を収集し，並列化・最適化を行う際のヒントを獲得し，並列化・最適化を施す系統的方法の確立を目指す．

参考文献

[1] A. V. Aho, R. Sethi, and J. D. Ulman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

[2] U. Banerjee, *Loop Transformations for Restructuring Compiles: The Foundations*, Kluwer Academic Publishers, 1993.

[3] M. Sasakura, K. Joe, Y. Kunieda, and K. Araki, "NaraView: An Interactive 3D Visualization System for Parallelization of Programs, " *International Journal of Parallel Programing*, Vol.27, No.2, pp.111–129, 1999.

[4] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler Transformations for High-Performance Computing, " *ACM Computing Surveys*, Vol. 26, No. 4, pp. 345–420, December 1994.