並列化コンパイラにおけるループ半周シフト変換の研究

5W−04

草場大我　岩澤京子
拓殖大学工学部情報工学科

1．はじめに

ループ繰り返しの並列化を行う際，ループ繰越し依存があるループの場合には，同期や通信が用いられるが実行効率の向上はあまり期待できない．このようなループの中には，loop-alignment（ループ半周シフト）変換で依存関係を解消することによって同期や通信を排除できるものが存在する．

本研究では，ループ内のループ繰越し依存を，ループ独立依存へと変換する，ループ半周シフト変換の適用条件を一般化し，ループ変換を実装することを試みた．

2．システムの機能

本システムは，機能限定したC言語のソースプログラムを入力として，ループ並列化を行いOpenMP指示文(#pragma omp parallel for ...)の埋め込まれたソースプログラムを出力する．

データフロー解析ではフロー依存を次の3つに分類する．

(a) ループ独立な依存

(b) 連続するループ繰越し依存（「連続する」とは常にバックエッジを1回通ることを表す）

(c) それ以外の依存

3．変換の適用条件

ループ中での一般的な文の移動を可能にすることにより，適用範囲を拡張するために，変数の名称変換を行っている．

A Study on loop alignment for parallelzing compiler
Taiga Kusaba, Kyoko Iwasawa
Department of Computer Science, Faculty of Engineering, Takushoku University

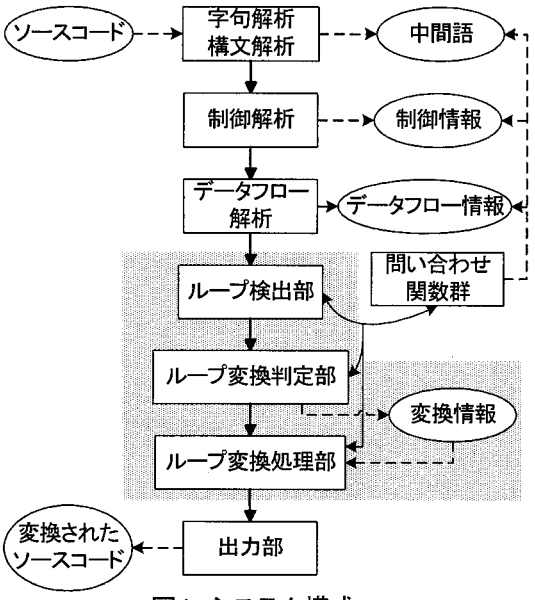

図1　システム構成

適用条件は次の2つである．

(1) ループ内のすべてのループ繰越し依存は連続し，かつ定義文はループ出口を支配している．

(2) ある変数に関するフロー依存の定義文と使用文の間には，他の変数に関するフロー依存関係はない．

生成するプログラムの可読性の保持のため，文を移動しない範囲での判定を行うこともできる．

4．ループ変換手順

ループ変換を行うプログラムの変換手順について述べる．

4．1　ループ変換候補となるループの選択

ループ変換の対象となるforループを抽出する．ループ繰越しフロー依存について2章の分類を行い，(c) が検出されたループを対象から外す．

4．2　変換の適用判定

3章の条件の成立可否を次の手順で調べる．

Step1 制御情報から条件(1)を満たすか調べる．

Step2 フロー依存と内部ループと条件構造から分割点を決めるためのグループ化を行う．次の3つの原則に従うグループ化の可否を判断する．

- 繰越し依存の定義点と使用点は別のグループにする．
- 独立依存の定義点と使用点は同じグループにする．
- 条件構造や内部ループは構造単位で同じグループにする．

この原則に従うグループ化ができない場合，半周シフトによる効果がないことがわかる．図2の例で，グループ①はAの独立依存と条件構造をまとめ，グループ②はBの独立依存をまとめる．このときBの繰越し依存の定義点と使用点は，原則通りグループ①，②に分かれる．

Step3 文の配置と分割点を決定する．グループを，繰越し依存の定義文と使用文の有無で3つに分類する．そして使用文を含むグループをループの先頭へ順に引き上げていく．引き上げた文の最後が半周シフト変換のための分割点である．　図2から図3のようになる．この例ではグループ①に属するAの使用を含む条件構造を，Bの定義文の前に移動し，この直後が分割点となる．

4．3　ループ変換処理

決定した分割点や文の配置の情報から中間語の書き換えてループ半周シフト変換を行う．同時に，対象となったループの前に OpenMP 指示文を挿入する．図2の入力ソースは図5のようになる．多重ループの場合でも，最内側ループから順次変換を行うことで対応できる．

5．おわりに

グループ化して文を移動することにより，条件分岐を含むループや多重ループにも半周シフト変換が適用できることを確認した．現在実装したのは変数のみだが，2章で分類したフロー情報が得られれば，配列参照のあるループの半周シフト変換も同様の手法で行うことができる．今後は，そのようなループにも対応できるように進めていく予定である．

参考文献
[1] A. V. エイホ 著，原田賢一 訳，コンパイラⅡ－原理・技法・ツール－，サイエンス社，1990
[2] 笠原博徳 著，並列処理技術，コロナ社，1991
[3] 新名博 田村智洋 藤原優史，手続き間解析を用いたグローバル変数のフローセンシティブな依存の解析方式，東京農工大学卒業論文，１９９７

```
main(){
  int A,B,C,D,E,F;
  int i;
  B = 0;
  for(i=0;i<100;i++){
    A = i + B;
    B = i - 2;
    if(i < 50){
      C = A * i;
      D = C + i;
    }
    E = i + 5;
    F = B / 5;
  }
}
```

変換前の入力例

```
A = i + B; ①
B = i - 2; ②
if(i < 50){
  C = A * i; ①
  D = i + 3; ①
}
E = i + 5; ③
F = B / 5; ②
```

図２　フロー依存によるグループ化

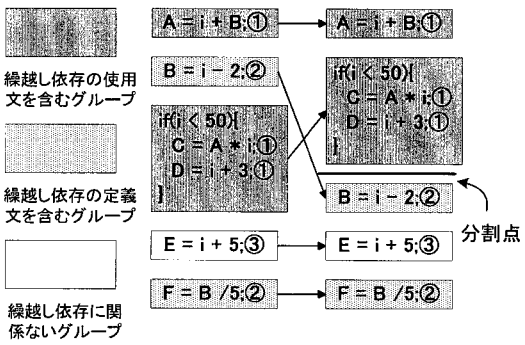

図3　文の配置と分割点

```
main(){
  int A,B,C,D,E,F;
  int i;
  int #A_10,#B_10,#C_10;
  B = 0;
  i = 0;
  A = i + B;
  if(i < 50){
    C = A * i;
    D = C + i;
  }
  #pragma omp parallel for
    private(#A_10,#B_10,#C_10,D,E,F) lastprivate(i)
    for(i=1;i<100;i++){
      #B_10 = (i-1) - 2;
      E = (i-1) + 5;
      F = #B_10 / 5;
      #A_10 = i + #B_10;
      if(i < 50){
        #C_10 = #A_10 * i;
        D = #C_10 + i;
      }
    }
  B = (i-1) - 2;
  E = (i-1) + 5;
  F = B / 5;
}
```

図４　変換後の出力例