はれだすにおけるループ間依存関係の記述方法とその適用例

3U－9

金子 正教　安江 俊明　萩原 純一　田原 歩　山名 早人　村岡 洋一
e-mail:harray-ml@muraoka.info.waseda.ac.jp
早稲田大学大学院 理工学研究科

1．はじめに

本稿では、マルチアーキテクチャコンパイラ開発環境－はれだす－[1][2]におけるループ間依存関係の記述方法とその適用例について述べる。

従来、プログラム中におけるデータ依存関係を特徴付ける方法して依存ベクトルが用いられている。しかし、この依存ベクトルは、同一ループ内あるいはループ外におけるデータ依存関係を記述するものであり、ループ間のデータ依存関係を特徴付けるための一般的な記述方法は定義されていなかった。これに対して本稿では、ループ間依存ベクトルを定義し、ループ間のデータ依存関係を記述する方法について述べる。また、ループ間依存ベクトルを用いることにより、ループ融合可能判定が従来手法に比べて容易に行えることを示し、さらに、各ループの並列性を失うことなくループを融合するためのループ間依存ベクトルの適用法について述べる。

2．ループ間依存ベクトル

ループL_1, L_2間のデータ依存は、L_1で定義または参照される変数と、L_2で定義または参照される変数とが同一のメモリをアクセスすることによって生じる。それらが、ループインデックス変数によって添字付けされた配列変数である場合には、L_1のあるイタレーションi_{L1}からL_2のあるイタレーションi_{L2}への依存となる。本節では、ループ間のイタレーションにまたがったデータ依存関係の記述方法としてループ間依存ベクトルを定義し、その算出方法を示す。

2.1 ループ間依存ベクトルの定義

ループ内で使用される配列変数の添字はループインデックス変数の線形結合で表されているものと仮定し、同一ネストレベルにある2つの隣接するDOループL_1, L_2間の依存ベクトル$V(L_1, L_2)$を以下のように定義する。

L_1のイタレーションiとL_2のイタレーションjとの間にデータ依存関係があるとき、$j=\alpha i+\beta$を満たすような(α, β)によって、

$$V(L_1, L_2)=(\alpha, \beta)$$

を定義する。ただし、jがiによらない任意のイタレーションであるときは、

$$V(L_1, L_2)=(0, *)$$

で表す。また、データ依存関係が存在しない場合には、

$$V(L_1, L_2)=\phi$$

で表す。一般には、ループ間に複数のデータ依存関係が存在するので、

$$V(L_1, L_2)=\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \cdots\}$$

のようなベクトルの集合になる。

A Description and its Application of the Data Dependence between Loops for -HAREDAS-

Masanori KANEKO, Toshiaki YASUE, Junichi HAGIWARA, Ayumu TAHARA, Hayato YAMANA, Yoichi MURAOKA
Graduate School of Science and Engineering, Waseda University

2.2 ループ間依存ベクトルの算出方法

ループ間依存ベクトルはループにまたがった1組の配列変数による依存に対して1つ定義される。図1(a)のループL_1, L_2間においては、A(aI+b)と A(cI+d)との間の依存に対して定義され、以下のようにして算出される。

L_1のイタレーションiにおけるA(aI+b)と、L_2のイタレーションjにおけるA(cI+d)との間の依存方程式を立て、

$$ai+b = cj+d$$

jについて解くと、

$$j = \frac{a}{c}i + \frac{b-d}{c} \qquad (2.1)$$

したがって、

$$V(L_1, L_2) = \left(\frac{a}{c}, \frac{b-d}{c}\right)$$

ただし、i, jはともに整数であるから、式(2.1)の右辺が整数にならないときはこのベクトルはデータ依存関係を表していない。例えば、図1(b)のループ間における依存ベクトルは、

$$V(L_1, L_2) = \left(1, -\frac{1}{2}\right)$$

となるが、このとき、

$$j = i - \frac{1}{2}$$

は整数になり得ないので、ループ間の依存関係は存在せず、

$$V(L_1, L_2) = \phi$$

である。

```
L1┌ DO I=1,N                L1┌ DO I=1,N
  └   A(aI+b)=…               └   A(2I-1)=…

L2┌ DO I=1,N                L2┌ DO I=1,N
  └   …=A(cI+d)               └   A(2I)=…
```

(a)　(b)

図1 ループ間依存ベクトルの算出例

3．ループ間依存ベクトルの利用

本節では、同一ネストレベルにある2つの隣接するDOループL_1, L_2間の依存ベクトル$V(L_1, L_2)$をループ間並列化可能判定およびループ融合可能判定に利用する方法について述べる。

3.1 ループ間並列化可能判定

$V(L_1, L_2)=\phi$ならば、L_1とL_2は並列に実行可能である。また、L_1とL_2の実行順序は任意である。例えば、図1(b)の2つのループは、$V(L_1, L_2)=\phi$であるからループ間の並列実行が可能である。

3.2 ループ融合可能判定

従来、ループ融合は2つのループ間にデータ依存関係が存在しないか、あるいは、融合後にデータ依存関係が保存されることを条件として判定されていた。本節では、ループ間依存ベクトルを用いることにより、この判定が行えることを示す。また、各ループの並列性を失うことなく融合するための依存ベクトルの適用法について述べる。

3.2.1 従来の融合可能条件[3][4][5]

2つの隣接するDOループL_1, L_2が融合できるのは次の(1)から(3)の条件を全て満たす場合である。ただし、L_1がL_2に先行するものとする。

(1)同一制御条件下で実行される．すなわち、L_1が実行されるときは必ずL_2も実行される．
(2)融合後もL_1からL_2へのデータ依存が保存され、融合により、L_2からL_1へのループ運搬依存が生じない．
(3)同一回数実行される．すなわち、ループ繰り返し回数が等しい．

以上のように従来のループ融合可能条件は、融合前の依存ベクトルのみから判定できるのではなく、融合したときに、依存ベクトルがどのようになるかを求めることによって判定している。

3.2.2 ループ間依存ベクトルを用いたループ融合可能判定

前節で述べたように、従来のループ融合可能判定は、判定時に融合後の依存ベクトルを計算する必要があった。これに対して、本節では、データ依存解析時に求められるループ依存ベクトルを用いて条件(2)を判定する方法を示す。

ループ間依存ベクトルを用いたときの条件(2)は、次の(2')のようになる。ただし、ループ範囲内で$\alpha i+\beta$の値が整数となるときのiの最小値をM，最大値をNとする。

(2')$V(L_1,L_2)$に含まれる各ベクトル(α,β)に対して
$\alpha \geqq 1$ かつ $\alpha M+\beta \geqq M$
または、
$\alpha < 1$ かつ $\alpha N+\beta \geqq N$
を満たす．

以上は、融合後にL_1とL_2の依存のあるイタレーションの実行順序が変わらないため、すなわち、L_1からL_2へのデータ依存が保存されるための必要十分条件である。L_1とL_2の依存のあるイタレーションの実行順序を変えなければ、L_2からL_1へのループ運搬依存は生じない。

[ループ間依存ベクトルを用いたループ融合可能判定の例]

```
L1┌ DO I=2,N                 L1┌ DO I=2,N
  └   A(I)=B(I)+C(I)            └   A(I)=B(I)+C(I)

L2┌ DO I=2,N                 L2┌ DO I=2,N
  └   D(I)=A(I-1)               └   D(I)=A(I+1)
      (a)                           (b)
```

図２ ループ融合可能判定の例

図２(a)のループ間依存ベクトルは、
$V(L_1,L_2)=(1,1)$であり、$1\cdot 2+1 \geqq 2$であるから、融合可能。

図２(b)のループ間依存ベクトルは、
$V(L_1,L_2)=(1,-1)$であり、$1\cdot 2-1 \not\geqq 2$であるから、融合不可能。

3.2.3 ループ融合時におけるループ間依存ベクトルの利用

ループ融合の利点は、一般にループオーバヘッドの削減とデータのローカリティを高めることにある[3][5]。しかし、融合することよって各ループの並列性を失う可能性がある。本節では、ループ間依存ベクトルを用いて、各ループの並列性を失うことなく融合する方法について述べる。

ループ間依存ベクトル$V(L_1,L_2)=(\alpha,\beta)$において、$\beta \geqq 1$のときにはそのままループを融合すると各ループの並列性を失う可能性がある。例えば、図２(a)のループでは、L_1,L_2ともにDOALLループであるが、融合後はDOSERIALループとなる。しかし、$\alpha=1$の場合には、L_1,L_2の繰り返し範囲をm～nとすると、L_2をβだけ後方にシフトしてm～n－βの範囲を融合することにより、融合後のループもDOALLループになる。m＝２，n＝Nの例を図３に示す。

```
L1┌ DO I=2,N                          A(N)=B(N)+C(N)
  └   A(I)=B(I)+C(I)                 ┌ DO I=2,N-1
                              ⇒      │   A(I)=B(I)+C(I)
L2┌ DO I=2,N                        └   D(I+1)=A(I)
  └   D(I)=A(I-1)                    D(2)=A(1)
V(L1,L2)=(1,1)
```

図３ ループ間依存ベクトルを用いたループ融合の例

また、従来の判定条件においては融合不可能と判定されるループに対しても、$\alpha=1$の場合には、L_2を$-\beta$だけ前方にシフトすることにより、m－β～nの範囲を融合することができる。m＝２，n＝Nの例を図４に示す。

以上をまとめると表１のようになる。

```
L1┌ DO I=2,N                          A(2)=B(2)+C(2)
  └   A(I)=B(I)+C(I)                 ┌ DO I=3,N
                              ⇒      │   A(I)=B(I)+C(I)
L2┌ DO I=2,N                        └   D(I-1)=A(I)
  └   D(I)=A(I+1)                    D(N)=A(N+1)
V(L1,L2)=(1,-1)
```

図４ ループ間依存ベクトルを用いた部分的な融合の例

表１ ループ間依存ベクトルによるループ融合

条件	融合方法
$\alpha=1$，$\beta \geqq 1$	L_2をβだけ後方にシフトして、m～n－βの範囲を融合する．
上記以外で(2')を満たす	そのまま融合する．
(2')を満たさず、$\alpha=1$	L_2を$-\beta$だけ前方にシフトして、m－β～nの範囲を融合する．

4．おわりに

本稿では、－はれだす－におけるループ間依存関係の記述方法とその適用例について述べた。ループ間依存ベクトルを用いることにより、ループ融合可能判定が従来手法に比べて容易に行える。また、各ループの並列性を失うことなく融合するため、および、従来の判定方法では融合不可能となるループを融合するためにループ間依存ベクトルを適用できることを示した。

現在、本手法を－はれだす－のデータ依存解析部および並列性解析部としてインプリメント中である。

今後の課題としては、3.2.3節において、$\alpha \geqq 2$の場合、および、ループ間依存ベクトルが複数存在する場合のその利用法があげられる。

謝辞

本研究の遂行にあたり、数々の助言をいただいた村岡研究室の皆様に感謝致します。

参考文献

[1] 安江ほか:“超並列のためのマルチアーキテクチャコンパイラ開発環境－はれだす－”,並列処理シンポジウムJSPP'92,pp.139-146,Jun.1992
[2] 安江ほか:“超並列のためのマルチアーキテクチャコンパイラ開発環境はれだすにおける内部表現”,第45回情処全大,3U-08,Oct.1992
[3] J.Ferrante et.al:“The Program Dependence Graph and Its Use in Optimization,” ACM Trans.on Prog. and Lang.Systems,Vol.9,No.3,pp.319-349,July.1987
[4] J.Warren:“A Hierarchical Basis for Reordering Transformations,” Conf.11th ACM Symposium on Principles of Programming Languages,pp.272-282,Jan.1984
[5] D.J.Kuck et.al:“Dependence Graphs and Compiler Optimizations,” Conf.8th ACM Symposium on Principles of Programming Languages,pp.207-218,Jan.1981