1L－10　グラフの変形に基づくソフトウェア・パイプライン化の方法*

小池祐二†　渋沢進†

茨城大学工学部情報工学科‡

1　はじめに

プログラムのループ部分をソフトウェア・パイプライン化することで、オブジェクトコードのサイズをあまり増大させること無く並列性を上げることができる[1][2][3]。また、データフローグラフ (DFG) はデータの依存関係を自然な形で表現できるため、並列性の抽出などの並列マシンのための最適化コンパイラにとって不可欠な処理の役に立つ。

そこで本稿では、データフローグラフに表されたループ部分のプログラムを、そのDFGを変形することでプロローグ、定常ループ、エピローグ部分に再構成する手順を述べる。グラフ変形の処理としてソフトウェア・パイプライニングが行えることで、逐次的 (非パイプライン的) なループのパイプライン化の自動化を容易にすることができる。

2　プログラムのグラフ表現

ソースプログラム上での各文をデータフローグラフに表し、求まった各グラフをデータの依存関係に従って1つのデータフローグラフにまとめていく。このデータフローグラフを求める過程で、無駄な命令、冗長な命令は可能な限り取り除いておく。このようにしてでき上がったデータフローグラフの中でも、特に次のような性質のノードに注目する。

- 入力ノード：アークが入らないノード (代入文の右辺に現れる変数)
- 出力ノード：アークを出さないノード (代入文の左辺に現れる変数)

サンプルプログラムを図1に、それをデータフローグラフに表したものを図2に示す。実際には前処理部、後処理部がこれに加わる。また、グラフを簡単にするために配列はインデックス計算、アドレス計算の部分をまとめて1つのノードで表した。

```
for(i = 1; i < N; i++){
    a[i] = i * a[i-1];
    b[i] = a[i] * b[i-1];
    c[i] = b[i] * c[i-1];
}
```

図1. サンプルプログラム

*A Method for Software pipelining Based on Graph Transformation

†Yuji KOIKE, Susumu SHIBUSAWA

‡Ibaraki University, Hitachi, Ibaraki 316, Japan

3　ソフトウェア・パイプライン化の手順

データフローグラフが与えられたとき、それを並列性の高い定常ループ、前処理部のプロローグ、後処理部のエピローグの3つの部分に切り分ける方法について説明する。

この方法は大きく分けて次の3ステップからなる。

1. 異なるイタレーション (ループ・ボディの実行) 間でのデータ依存解析
2. DFGの色分けによるステージへの分割
3. プロローグ、定常ループ、エピローグの組み立て

このようにして定常ループが求められた後、定常ループ内の命令をスケジューリングする。

基本的な方針は、異なるループにまたがるデータの依存関係にしたがってループ・ボディを複数のパイプライン・ステージに分割し、ステージ単位でループを再構成することである。次に,上の各ステップを図1のプログラムを例にやや詳しく見ていく。

[ステップ 1]　$1 \leq I \leq N-1$ とし、ループの制御変数 i について、$i = I$ のときの出力ノードの集合を $V_{out}(I)$、$i = I+1$ のときの入力ノードの集合を $V_{in}(I+1)$ と表す。また、k回後のイタレーションとの間でフロー依存関係にあるノードの集合を $V_{flow}(k) = V_{out}(I) \cap V_{in}(I+k)$, $(k \geq 1)$ と表す。

例えば、図1のプログラムでは次のようになる。

$$
\begin{aligned}
V_{out}(I) &= \{a[I], b[I], c[I]\} \\
V_{in}(I+1) &= \{i, a[I], a[I+1], b[I], b[I+1], c[I]\} \\
V_{flow}(1) &= \{a[I], b[I], c[I]\}
\end{aligned}
$$

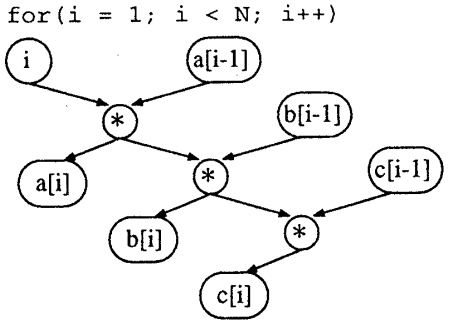

図2. データフローグラフ

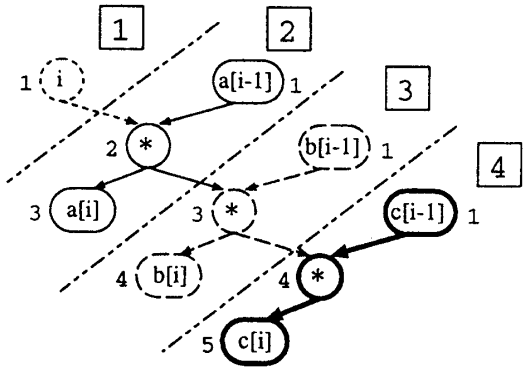

図 3. DFG の色付けとステージへの分割

[ステップ 2] i) DFG の上流のノードから深さをつけていく。図 3 では各ノードの横に書いてある数字が深さを表す。ii) 各出力ノードを深さの大きい順に選び、次のように色づけする。選ばれた出力ノードとフロー依存の関係にある入力ノードから到達できる全てのまだ色づけされてないノードを色づけする。色を変えてこれを繰り返す。図 3 では 4 色で色分けされた。iii) 各色で塗られた部分グラフをそれぞれ 1 つのステージとし、それを 1 つのノードで表し、データフローグラフを作る。これをステージフローグラフと呼ぶことにする。図 3 では、四角で囲んだ数字がステージ番号を表す。iv) ステージフローグラフが一方向かつ一直線となるようにステージの併合を行う。この例では必要ない。

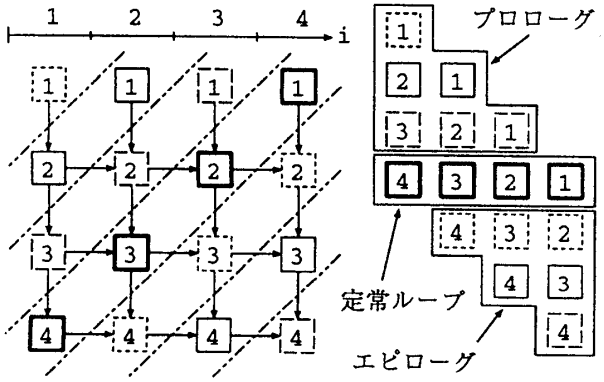

図 4. ステージ間の依存関係とループの再構成

[ステップ 3] i) ステージフローグラフをステージの数だけ並べ、ステップ 1 で求めた結果にしたがってイタレーション間でのデータ依存関係をアークで書き加える。ii) このグラフを、各ノードの実行開始可能時刻ごとに色分けし、同じ色で塗られたステージの数が最も多いところを定常ループとする。その他の部分がプロローグ、エピローグとなる。図 4 は、横軸がループのイタレーションを、各列がループ・ボディを表す。アークはステージ間の依存関係を表し、この依存関係にしたがってステージを色分けしていく。すると、太線で表したような定常ループが見つかる。

図 5 にこの方法で求められた定常ループを示す。各配列のインデックスはこの図のように変更する。ステージ 1 はループ制御変数の更新のときに実行されるので、この図には示してない。REGx と表されているノードは、前後のイタレーションとの間でレジスタを介してデータの受渡しをすることを表す。図 5 には、値が読み出された後でないと書き込みができないという逆依存関係 (write after read) が、レジスタノード間にいくつかみられるが、これによって並列度があまり上がらないときにはレジスタを一時的にリネームすればよい。

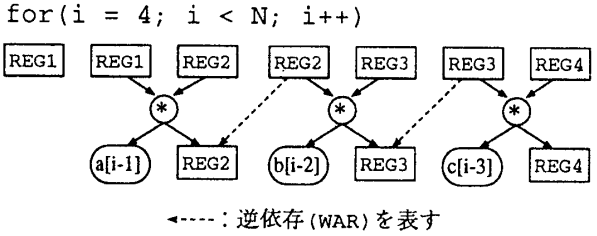

図 5. 求められた定常ループ

4 本方法によるソフトウェア・パイプライニングの効果

本手法によって求められた定常ループを、パイプライン化していないループ・ボディの実行時間と比較した。想定した細粒度並列計算機は整数ユニット ×2、不動小数点ユニット、ロードユニット、ストアユニットがそれぞれ 1 つとした。ロード命令、ストア命令の実行時間はそれぞれ 3MC(マシンサイクル) と 1MC、整数の加減算命令、乗算命令はそれぞれ 1MC と 3MC、分岐命令は 2MC とした。

図 1 のプログラムの場合、ループ 1 回の繰り返しにかかる時間は、パイプライン化した場合は 9MC、パイプライン化してない場合は 11MC であり、18 %程度高速化されている。

5 おわりに

本方法では配列に規則的にアクセスするようなループにソフトウェア・パイプライニングを施す方法について述べてきた。また、連続するイタレーション間でデータ依存があるようなデータ依存が密に存在する場合については、その制約の中でできる限り並列性の高いソフトウェア・パイプラインを求めることができた。

今後の課題は、依存性が低く、比較的自由にループの再構成ができるような場合、そして、ループ内に if 文を含む場合についても、並列性の高いソフトウェア・パイプラインが求まるようにすることである。

参考文献

[1] 斎藤光男, 井上淳 : RISC 系の細粒度並列マシン, 情報処理, Vol.35, No.10, pp.940-951 (Oct. 1994).

[2] 井上和彰 : スーパースカラ・プロセサの性能を最大限に引き出すコンパイラ技術, 日経エレクトロニクス, No.521, pp.165-185 (Mar. 1991).

[3] 中澤喜三郎 : 計算機アーキテクチャと構成方式, 朝倉書店, p.570 (1995).