ループ並列化における同期位置決定方法

6Y-3

岡部　公治　　浅原　重夫

松下電器産業株式会社　情報システム研究所

1．まえがき

近年、マルチプロセッサの商用化が進むにつれ、単一のプログラムを分割して並列処理する試みが行われているが、その際、従来の逐次型言語を並列処理用に変換する技術が必要となる。

並列処理されるプロセス間でデータ依存関係が存在する場合、実行結果を保証するためにそれらのプロセス間での同期が必要である。本稿では、プログラムのフローが複雑な場合、同期処理の挿入位置の決定方法を提案する。

本手法によれば、同期位置の決定が比較的簡単であり、冗長な同期命令の除去も容易である。

2．並列化と同期処理

2．1　ループ並列化

並列処理の方法に関しては、以下を仮定する。

(1) プログラムの繰り返し構造(以下ループと呼ぶ)(FORTRANのDOループやIF～GOTOによるループ)を分割し、複数のプロセスで処理する。

(2) ループは複数の文から構成され、文は通常の実行文以外に、条件判定文(IF,THEN,ELSE)や分岐文(GOTO)を含む。(図1)

(3) ループ内で用いられるデータが、異なる繰り返しのループ間で依存関係を持つ場合、プロセス間で図2で示されるような同期処理が必要となる。

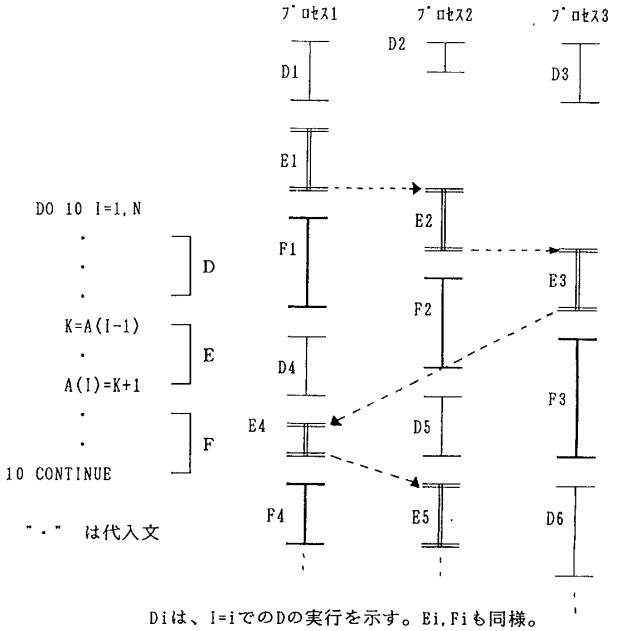
```
⓪        DO 10 I=1,N
①           K=2+L
②           DO 20 J=1,N
③              B(J)=A(I,J)+B(J-1)
④     20       A(I,J)=I+J
⑤           IF(K.EQ.2) THEN
⑥              C(I)=I
⑦           ELSE
⑧              C(I+2)= -I
⑨           ENDIF
⑩           IF(D(I).DE.0)  GOTO 30
⑪           D(I)=D(I)-1
⑫     30    C(I)=A(I,C(I+1))+D(I+1)
⑬     10 CONTINUE
```

図1　条件判定・分岐を含むDOループ

2．2　同期処理位置の決定

プロセス間の同期を、同期処理用関数を利用して実現する場合、プログラム中のどの位置にそれらの関数を用いるか決定しなければならない。条件判定文や分岐文を含まない場合（図2）は簡単であるが、含む場合（図1）になんらかの決定方法が必要である。

例えば、図2では、Eの部分が逐次処理されるように、Eの直前にWAIT、直後にSIGNALといった同期処理用関数を挿入すればよい。図1のDOループで文⑫の実行は、文⑧に対してその1ループ前の終了に先立ってはいけない。そのため文⑫の直前にWAIT命令を挿入するがSIGNAL命令は文⑧の直後だけに挿入したのでは、それが確実に実行される保証ができないので、文⑤の直後にもSIGNAL命令を挿入するか、あるいはこのＩＦブロックの終了後、文⑨の直後にSIGNAL命令を挿入する必要がある。



Diは、I=iでのDの実行を示す。Ei,Fiも同様。

図2　同期を伴う並列処理

3．同期化アルゴリズム

本手法を簡単に述べると、分岐の毎に同期命令を実行するのではなく、プログラム中の確実に実行される位置に挿入する方法である。

A method for determination of synchronization points in loop parallelization
Koji Okabe , Shigeo Asahara
Matsushita Electric Industrial Co.,Ltd

並列化対象ループに対し、次の４ステップで同期処理位置を決定する。以下では、ループ内への飛び込みやループ外への飛び出しはないものと仮定しているが、それらが存在する場合にも、容易に対処できる。

(1) フローグラフの作成

プログラムの分岐・条件判定を考慮し、各ループに対してフローグラフを作成する。（図３は図１のループに対応）

(2) 深さ優先探索

フローグラフを深さ優先探索し 、各文nのDFNUMBER[n],LOW[n]を求める。[1]

DFNUMBER[n]:文nの訪問順番

LOW[n]:文nからbackedge（深さ優先探索で、最終文まで到達した時に通過しなかった枝）を辿って戻れる文のDFNUMBERのうち最小のもの

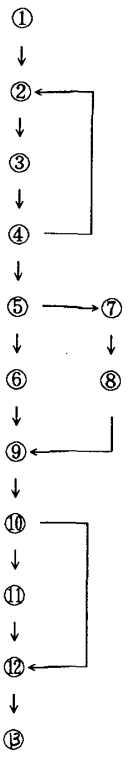


図３　フローグラフ

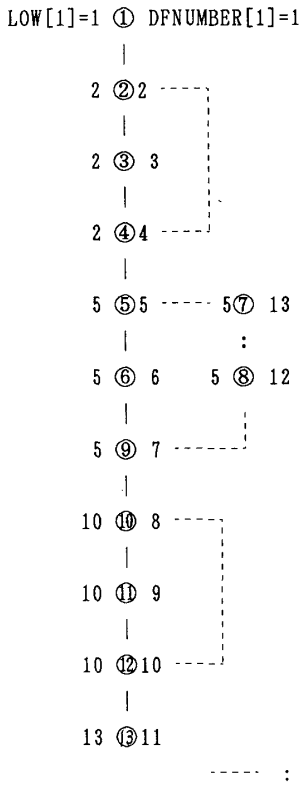


図４　深さ優先探索

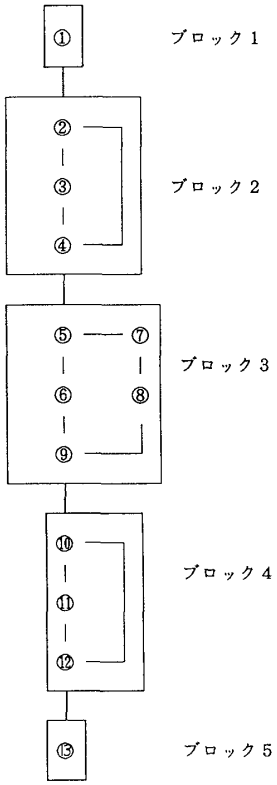


図５　ブロック分割図

図４に、図１のループに対して深さ優先探索を行った場合を示す。

(3) ブロック分割

フローグラフを図５で示されるように、入出力枝を唯一つ持つようなブロックに分割する。DFNUMBER[n]=LOW[n]　なる文nは、backedgeを持たず、深さ優先探索で発見される２連結成分の中で最小のDFNUMBERを持つ文である。これを各ブロックの先頭文とみなせば、仮定より、ループの先頭文は第１ブロックに、最終文は最終ブロックに属するので、このようなブロックへの分割が可能である。

```
begin
  for j=m,m-1,･･･,1
     for i=1,2,･･･,m
        begin
           if SYN(i,j)=1 then do
              for k=1,･･･,j
                 for l=i,･･･,m
                    SYN(l,k)=0
        end
end
```

図６　冗長な同期の除去手順

(4) 同期処理位置の決定

２つの文がデータ依存関係を持つ場合、先に実行すべき文を含むブロックｉの直後と、後で実行すべき文を含むブロックｊの直前が、同期処理用関数を挿入すべき位置であり、ＳＹＮ（ｉ，ｊ）＝１とする。

(5) 冗長な同期の除去

ループ内のブロック数をｍとすると、図６のアルゴリズムで冗長な同期を除去できる。

図５の例では、ブロック４→３間の同期が存在するので、ブロック４→２間の同期は不必要となる。

４．むすび

本手法は、次の２つの特長を有する。

(1) 分岐の毎に同期処理用関数を挿入するのに比べ、少ない計算量（文の数のオーダー）で決定することができる。

(2) 冗長な同期の除去が容易。

並列処理のソフトウェアの研究・開発は重要な分野であり、本手法は並列処理プログラムの作成、自動並列化コンパイラ等への応用が考えられる。

［参考文献］

1）J.E.Hopcroft and J.D.Ullman : Design and Analysis of Algorithms,Bell Telephone Lab., 1974