1N-3

並列処理システム－晴－における マクロブロック終了検出

草野義博、萩原孝、山名早人、村岡洋一
(早稲田大学　理工学部)

1．はじめに

我々が提案している科学技術計算処理用データフロー・マルチプロセッサシステム－晴－では、各プロセッサエレメントへ割り当てるタスクの分割にマクロブロックという概念を用いている[1]。マクロブロックとはプログラムをある基準に従って図1のように分割したもので、－晴－ではマクロブロックを単位としてプロセッサエレメントにタスクを割り当てる。マクロブロック内部ではデータ駆動制御で計算を進めて自然に並列性を抽出し、さらにマクロブロック間にコントロールフロー制御を導入し階層的な制御構造をとる。このような方法により、制御命令の増加などのデータ駆動制御の欠点を補うことができる[2]。

しかし、マクロブロックを単位としてタスクを割当てる際に種々の問題が生じる。マクロブロックの終了検出を高速に行なう必要があることもその一つである。そこで、本稿ではマクロブロックの終了検出を高速に行なう手法について述べ、簡単な評価を行なう。

2．終了検出のオーバーヘッド

プロセッサエレメントへのタスクの割り当てを、手続き（関数、サブルーチン等）のようにそれ自身が閉じているプログラム単位でおこなう場合には、タスクが出力する処理結果の個数は一般に少ない。したがって、タスクの終了つまりタスクの全ての出力が得られたか否かを検出するためのオーバヘッドはあまり問題とならない。これに対して、マクロブロックのようにプログラムの一部分を切り取ってきたようなものでは、

①タスクの出力の個数が多くなる傾向にあるため、タスクの終了を検出するためのオーバーヘッドが大きくなる。
②マクロブロックは手続きにくらべてプログラムの粒度が細かいため、タスクの個数が多くなりタスクの終了を検出する回数も多くなる。

等の問題点がある。特にマクロブロック内部にDOループが含まれている場合には、出力の個数はDOループの繰り返し回数に比例している場合が多いため、①が大きな問題点となる。

そこで－晴－ではタスク、つまりマクロブロックの終了を高速に検出するための機構が必要となる。

3．同期ノードを用いた終了検出

従来は、マクロブロックの終了検出には同期ノードを用いていた。同期ノードは2個の入力が揃えば発火して同期データを出力するものであり、このノードの目的は2個のデータが揃っているか否かを検出することである。

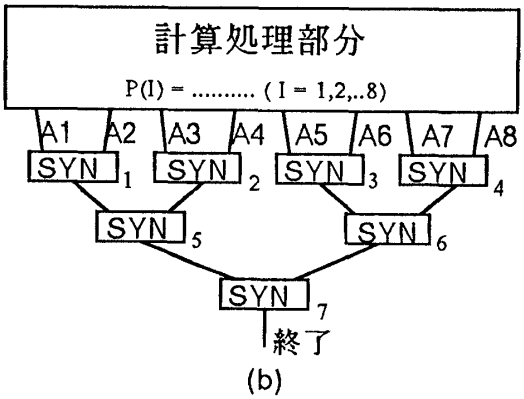
図2(a)に示すDOループのプログラムに対して、この同期ノードを用いた終了検出方法を同図(b)に示す。本来の計算処理の下にツリー形をした終了検出のためのプログラムが付加されている。計算処理部で配

```
      DO 10 I = 1, N
10    Q(I) = S + P(I)
      IF (S .NE.0.0) THEN
        DO 20 I = 1,N
20        Q(I) = Q(I) / S
      ELSE
        DO 30 I = 1,N
30        P(I) = 2.3 * P(I)
      ENDIF
      DO 40 I = 1,N
40      S = S + Q(I)
```

図1：マクロブロック化

```
      DO 10 I = 1, 8
10    P(I) = .....
```

(a)



(b)

図2：同期ノードを用いた終了検出

The Completion Detection of Macro-Block in Parallel Computer System -Harray-
Yoshihiro KUSANO,Takashi HAGIWARA,
Hayato YAMANA and Yoichi MURAOKA
School of Science and Engineering,Waseda Univ.

列Pの要素P(i)が計算されると、P(i)の計算終了を示すアクナレッジAiが出力される。Pの全ての要素が得られると、終了検出部の全てのノードが発火して、ツリーの根の部分にアクナレッジが現れ、終了が検出される。しかし、この方法では次のような問題がある。

①アクナレッジがツリーの根に進むにしたがって並列度が減少し、オーバーヘッドが増加する。
②アクナレッジが得られる順番が不確定の場合に、同期ノードが順番に発火できずオーバーヘッドはさらに大きくなる。

4．ー晴ーにおける終了検出

これらの点からー晴ーでは、DOループのように多数の出力があるマクロブロックの終了検出をオーバーヘッドが少なく行なえる特別な命令／機構を設ける。マクロブロックの終了＝「マクロブロックの全ての出力が得られた」と考え、出力データの個数をカウントする方式を取る。そのためのカウンタをハードウェアで用意し、またこのカウンタをインクリメントするカウント命令を用意する。この命令を図２(b)のDOループに適用した例を図３に示す。右入力には出力の個数（例では８）が入力され、左入力には計算が終了したことを示すアクナレッジA1~A8が入る。いずれかのアクナレッジが計算処理部から出力され、右入力である８とマッチングしてノードが発火すると、カウンタの値を１つインクリメントした後、カウンタの値とノードの右入力の値８とを比較する。カウンタの値の方が小さいときには何も行なわず。カウンタの値と右入力の値８が等しい場合には、全てのアクナレッジをカウントしたことになるので、データを出力してPの全ての要素が計算されたことを知らせる。この命令は図２(b)の終了検出のためのツリー形のプログラムと全く同じ働きをするものである。

5．比較

次に、この命令を用いた終了検出方法とツリー形の同期命令を用いた方法を簡単に比較してみる。比較のために、データフロー計算機のモデルとして図４のようなマッチングメモリと演算部からなるループを考える。仮定としてマッチングメモリの動作周期をｔとし、データがループを１周のにかかる時間をこの４倍とする。このモデルにおいて図２(a)のプログラムを実行させ、このときマッチングメモリで発火するノードに注目して、タイムチャートを描くと図５のようになる。同期ノードを用いる方法では、処理がツリーの根の部分に進んでゆくのに従って並列度が減少し、オーバーヘッドが増加していることがわかる。一方、この

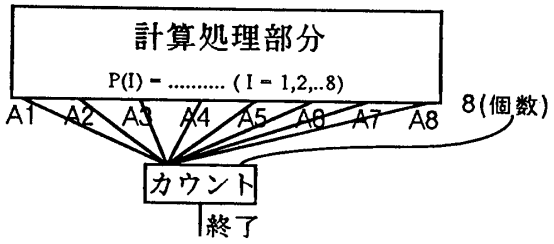


図３：カウント命令による終了検出

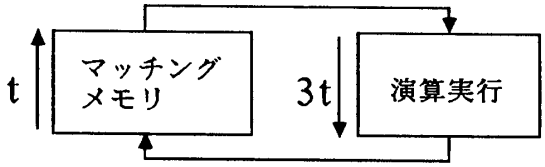


図４：データフロー計算モデル

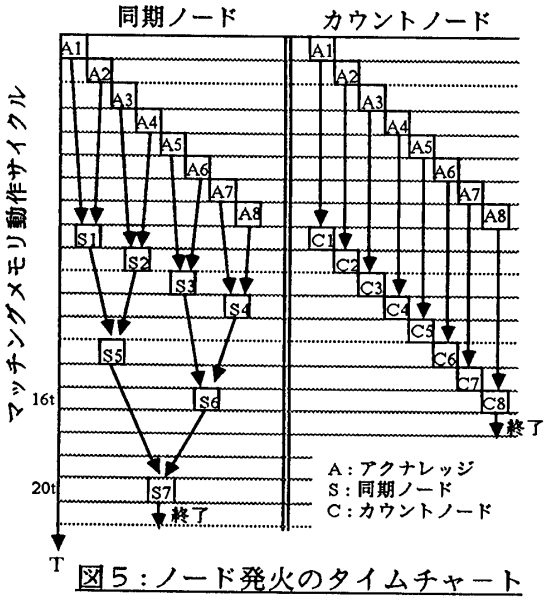


図５：ノード発火のタイムチャート

命令を使用することにより、多数の出力がある場合でもマクロブロックの終了を検出するためのオーバヘッドが小さいことがわかる。

6．おわりに

ー晴ーはその実行方式から、マクロブロックの終了を高速に検出する方法が必要である。本稿ではその方法について述べ、簡単な評価を行いその有効性を確かめた。

[1] 萩原他 ："並列処理システムー晴ーにおけるＦｏｒｔｒａｎプログラムのマクロブロック化の評価",情処第35回全体,1C-4,(1987)
[2] 丸島他 ："並列処理システムー晴ーの実行方式",情処研報,88-CA-69,pp.9-16,(1988)