6X-1 細粒度並列計算機ＮＬＩＴＨの概要

塩野賢二[1] 松永幸三[2] 小松秀昭[3] 深澤良彰[2] 門倉敏夫[2]
Kenji SHIONO Kozo MATSUNAGA Hideaki KOMATSU Yoshiaki FUKAZAWA Toshio KADOKURA

[1]三菱重工業㈱ MTSUBISHI HEAVY INDUSTRIES,LTD.
[2]早稲田大学 理工学部 Waseda University
[3]日本アイ・ビー・エム㈱ IBM Japan,Ltd.

１．はじめに

より高速にプログラムを実行したいという要求を満たすため、種々の並列計算機システムが研究・開発されてきた。システム・プログラム等の様に、陽には並列性を含んでいないプログラムにおいても、３～４命令程度の並列性を含んでおり[1]、これらを並列に実行させるアーキテクチャも、成果を上げつつある[2]。

我々が提案しているNLITH(N Lane Instruction THread computer)では、単一のプログラムを実行するために、複数のＰＥが各々独自な命令ストリームをもつ。この各命令ストリーム間では、同期を必要とする場合のみ、同期をとる。

２．NLITHの基本原理

NLITHでは、各PEはそれぞれ固有の命令ストリームを持っており、それぞれの命令ストリームを独立して実行する。同時に実行されなければならない命令をハードウェア・タグによって指定することにより、命令間の同期をとる。これによって、命令の種類により実行所要時間が異なる場合や、キャッシュ記憶を使用している場合などコンパイル時には実行所要時間が確定しない場合にも、無駄な実行待ち時間を短縮することができる。

また、NLITHでは同期をとるためのタイミング合せの無駄な命令を挿入する必要がないので、命令としてフェッチしなければならないデータの量を少なくすることができ、バスの利用率を抑えることができる。

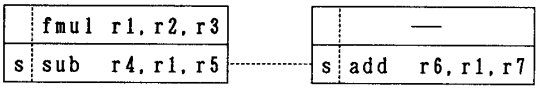
３．命令の制御機構

（１）ｓｙｎｃビット

例えば、図２(ａ)の様な命令列を２つのPEで実行させると図２(ｂ)の様になる。

図２中のタグｓをsyncビットと呼び、これによって指定された命令は、ハードウェアによって、必ず同時に実行されることが保証される。したがって、fmul命令の結果がr1に格納される前にadd命令が実行されることはない。

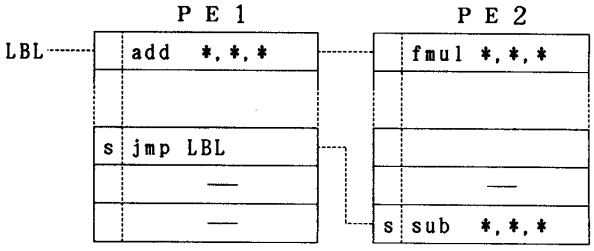
```
fmul r1,r2,r3
add  r4,r1,r5
sub  r6,r1,r7
```
(ａ)



(ｂ)

図２：ｓｙｎｃビットの例１

```
LBL1: add  *,*,*
      fmul *,*,*
        ⋮
      sub  *,*,*
      jmp  LBL1
```
(ａ)



(ｂ)

図３：ｓｙｎｃビットの例２

また図３(ａ)様な命令のシーケンスを２つのPEで実行すると、図３(ｂ)の様になる。

PE1がブランチ命令を実行すると、すべての

An Outline of Fine Grained Parallel Computer NLITH
Kenji SHIONO, Kozo MATSUNAGA, Hideaki KOMATSU, Yoshiaki FUKAZAWA, Toshio KADOKURA
MITSUBISHI HEAVY INDUSTRIES, Waseda University, IBM Japan

PEの制御がLBL1に移る。したがって、このプログラムを正しく実行するためには、ブランチ命令とsub命令を同期させる必要がある。

(2)マルチプル・ジャンプ

同時に複数のブランチ命令を実行するのが、マルチプル・ジャンプである。本来一つ一つ順次評価しなくてはならなかったブランチ命令が、1回の評価で済むようになる。

例えば、図4(a)の様な命令列があったとする。これをシーケンシャルに実行した場合には、最大で3サイクル必要である。しかし、これらの命令を、図4(b)の様に、3つのPEに割り当てて実行させると1サイクルで実行可能になる。

```
jeq r1,r2,LBL1
jeq r3,r4,LBL2
jeq r5,r6,LBL3
```

(a)

PE 1	PE 2	PE 3
jeq LBL1	jeq LBL2	jeq LBL3

(b)

図4：マルチプル・ジャンプ

この時、複数の条件が真になると、上に置かれている命令を実行した結果を優先させなければならない。

ここで、全部のPEが同じ論理アドレスにジャンプするので、アドレスを合せるためにnop命令を挿入する必要がある場合がある。例えば、図5の様に、LBLの位置にジャンプして来る場合を考える。

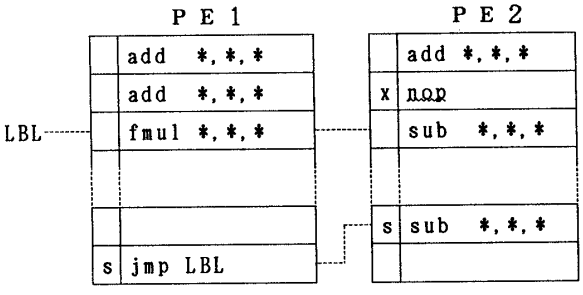


図5：skipビットの例

ブランチ命令がいずれかのPEで実行された場合、すべてのPEが同じ論理アドレスにジャンプする。したがって、論理アドレスを合せるために、図5中の波線で示すように、nop命令を挿入しなければならない。

このとき、この命令列を前から順に実行してきた場合、図5中の波線で示したnop命令は、ただ時間を浪費するだけである。この様な無駄をなくすために、タグ中に図5のxで示すskipビットを用意する。skipビットの立っている命令は、インストラクション・パイプラインには取り込まれず、次の命令がフェッチされる。

(3)部分同期

これまでは、すべてのPEが一斉に同期をとる(全同期方式と呼ぶ)ことを考えた[3]。しかし、実際には、全てのPEが同期を取る必要がない場合もあので、同期機構の発展形として、部分同期方式を導入する。これは、、特定のPE間のみでの同期を指定できる様にしたものである。

例えば、図6(a)の命令列は、部分同期を用いることにより、図6(b)の様に各PEに割り当てられる。

部分同期方式を導入することにより、無駄な同期待ちの状態を無くすことができ、より効率的な処理を期待できる。

```
add  r1,r2,r3
fmul r3,r4,r5
add  r3,r6,r7
fmul r7,r8,r9
sub  r7,r10,r11
sub  r5,r12,r13
```

(a)

PE 1		PE 2		PE 3	
	add r1,r2,r3	1	sub r7,r10,r11	1	fmul r3,r4,r5
3	add r3,r6,r7				sub r5,r12,r13
2	fmul r7,r8,r9				

(b)

図6：部分同期

4．参考文献

[1] H.Hagiwara, S.Tomita, S.Oyanagi, K.Shibayama："A Dynamically Microprogramable Computer with Low-Level Parallelism" IEEE Trans. on Comp., Vol. C-29, No.7, pp.577-595(1980)

[2]「特集：マイクロプログラミング」 情報処理, Vol.28 No.12 (1987)

[3] 松永、小松、深沢、門倉：「複数の命令ストリームを持つコンピュータのアーキテクチャ」信学技報, Vol.89, No.55, CPSY89-7 (1989)