FIFOキューを同期手段とする並列プログラムについて(I)†

——待ちなし並列プログラム——

有田 五次郎††

MIMD型並列処理機械は最も一般的な並列処理システムと考えられるが，プログラムの並列化の問題，メモリアクセス競合の問題の外に同期のオーバヘッドの問題があり，高多重並列処理は困難であるとされている．本論文ではまずMIMD型並列計算機における同期の問題について考察し，同期問題が，並列実行されるオペレーションの実行時間と同期操作の実行時間との相対的な問題であることを明らかにする．次に，オペレーションを点，制御の移動を枝とする木構造グラフで表現される待ちなし並列プログラム(SPP)の概念を示す．SPPは各プロセッサが先着順(FCFS)で各オペレーションを実行するとき，自然に同期がとれる並列プログラムになっている．FCFSはFIFOメモリを使用して簡単にハードウェアで実現でき，SPPはMIMD型並列処理において同期手段を高速化する一つの方法となる．SPPを実行する並列計算機は通常の機械語命令の外に並列分岐(parallel branch)命令およびタスク切換(exchange task)命令をもつ．ここではさらにこのような計算機のハードウェアの構成を示し，その上で動く幾つかの並列プログラムの例を示す．

1. はじめに

Flynn[1]の分類による多命令多データ(multi instruction stream multi data stream，以後MIMDと書く)型並列処理システムは，最も汎用性の高いシステムと考えられるが，多重プロセッサシステムの上では並列度を上げると同期の回数が増加し，同期のオーバヘッドによって並列化の効果が上がらなくなるとされている[2]．

多重プログラミングのオペレーティングシステム，チャネル結合・I/O結合の複合計算機システム，回線結合の計算機網など，従来のMIMD型並列処理システムでは同期手段をソフトウェアで用意している．並列実行するオペレーション(タスクまたはプロセス)が大きく，またその数もあまり多くないこれらのシステムではこれで十分効率のよい並列処理が行えるが，処理速度の向上を目的とする高多重並列処理システムでは並列実行するオペレーション(演算または操作)は小さく，また数も多いので，効率のよい並列処理を行うためにはソフトウェアによる同期手段だけではなく，ハードウェアによる高速な同期手段が必要になる．

ハードウェアによる簡単で高速な同期手段としてFIFOキューがあり，先行制御・パイプライン演算・通信回線制御等で同期を目的として使用されている．

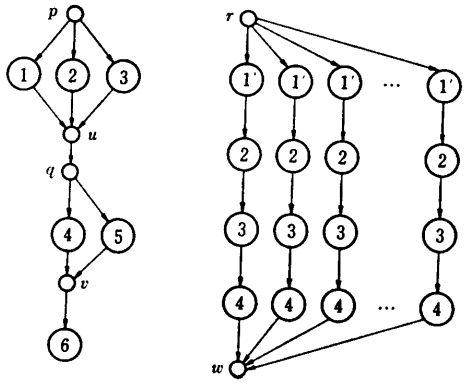
ここではFIFOキューを並列プログラムの同期手段とするため，待ちなし並列プログラム(SPP)の概念を導入し，これを実行するハードウェアについて考察する．SPPはデータの依存関係に基づいた木構造グラフで表現され，各オペレーションを割り当てられたプロセッサが逐次実行を行うと自然に同期がとれる並列プログラムとなっている．

並列プログラムに関する研究は数多くあり，種々の同期操作が提案されまた実現されている[3),4)]．これらの研究は主として並列プログラムの論理的性質や同期操作の能力の究明等を目的としており，並列プログラムの物理的性質や同期操作の実行効率等にはあまり考慮が払われていない．したがって実際のシステムでは，同期操作は割込み機能とTest & Set/Reset (Lock/Unlock)を使用して，ソフトウェアで実装されている．ここに提案するSPPは並列プログラムの物理的性質—オペレーションに対するプロセッサ割当て—を利用して同期手段を簡単化し，これをハードウェアで実装しようとするものである[5),6)]．

本論文は次の構成をとる．まず2章で非巡回定向グラフで表現される並列プログラムを考え，この上での同期のオーバヘッドについて考察する．3章では上述のSPPと実行管理のハードウェアの概要にについて述べる．システムはメモリ共有型高多重プロセッサシ

† On a Parallel Program with Synchronizing Mechanism Using FIFO Queue (I)—Self Synchronizing Parallel Program—by ITSUJIRO ARITA (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University).

†† 九州大学工学部情報工学科



(a) Program A. (b) Program B.

図 1 並列プログラム例
Fig. 1 Examples of parallel program.

ステムである．最後に4章でこのシステム上で実行される簡単なプログラム例を示し，同期のオーバヘッドを推定する．

2. 並列プログラムと同期問題

2.1 並列プログラム

逐次的プログラムはオペレーションの系列として表現される．ここでオペレーションは一つの演算または操作であってもよいし，幾つかのオペレーションから成るプログラムの断片（以後 P-片と呼ぶ）であってもよい．

プログラム中のオペレーションは，相互にデータ依存関係がない場合に並列実行が可能で，このような並列プログラムがオペレーションを点，制御の移動を枝とする定向グラフで表現できることはよく知られている[3]．たとえば次の二つのプログラムは，それぞれ**図1**(a), (b) のように表現できる．

プログラム A

① $SX=\mathrm{SIN}(X)$
② $SY=\mathrm{SIN}(Y)$
③ $CXY=\mathrm{COS}(X+Y)$
④ $Z1=CXY*(SX+SY)$
⑤ $Z2=\mathrm{SQRT}(SX*SX+SY*SY$
⑥ $Z=Z1/Z2$

プログラム B

① DO 10 $I=1,\ N$
② $X=PI*\mathrm{FLOAT}(I-1)/10.0$
③ $Y=PI*\mathrm{FLOAT}(N-I)/5.0$
④ 10 $Z(I)=F(X,\ Y)$

ここに，p, q, r および u, v, w はそれぞれ，並列動作を表現するため導入した並列分岐および同期のオペレーションで，(b) の⑪ は次の形をしている．

$I=i \quad (i=1, 2, \cdots, 10)$

図1において，同一列のオペレーションは同一プロセッサで，異なる列のオペレーションは異なるプロセッサで実行されるとすると，下方に向う枝は同一プロセッサ上での制御の移動—すなわち逐次的動作，横方向に向かう枝は異なるプロセッサ上のオペレーションへの制御の移動—すなわち並列動作となり，並列プログラムを表現している．

図1(a) および (b) の各点がプログラムAおよびBの各文を表しているとき，A，Bの実行結果と (a), (b) の実行結果が等しいことは容易に確かめられる．

ここではこのように，同一列のオペレーションが同一プロセッサで実行されるようなグラフで表現された並列プログラムを扱う．

図1の並列プログラムには条件分岐およびループが含まれていない．しかし，並列処理の対象になるプログラムには必ずループが含まれている．ループを構成する条件分岐および無条件分岐は同一プロセッサ上の制御の移動である．したがって一般の並列プログラムは，図1のような並列プログラムを条件分岐および無条件分岐で結合することによって得られる．

2.2 同期問題

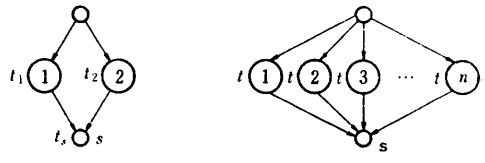
図2(a) は並列プログラムにおける最も簡単な同期モデルである．ここに t_1, t_2 はそれぞれオペレーション①，②の実行時間であり，t_s は同期操作 s を1回実行するのに要する時間である．

このような並列実行が意味をもつためには，t_1, t_2 および t_s 間に次の関係がなければならない．

$$t_1 \simeq t_2,\ t_1 > 2 \cdot t_s,\ t_2 > 2 \cdot t_s \qquad (2.1)$$

同期操作は本質的に逐次的である．したがって図2(b) のような形で実行時間 t の n 個のオペレーションを並列実行したとき，速度の向上率 N は次のようになる．

$$N=\frac{n \cdot t}{t+n \cdot t_s}=\frac{n}{1+n(t_s/t)} \qquad (2.2)$$



(a) Synchronization of two operations. (b) Synchronization of n operations.

図 2 同期モデル
Fig. 2 Models of synchronization of operations.

これより並列化の効果が上がるためには

$$t_s/t \ll 1 \qquad (2.3)$$

が必要である.

一つの処理を並列実行するとき，並列度を上げるためにはなるべく多くの独立なオペレーションに分割することが必要である．しかし処理を細分化すると個々のオペレーションの実行時間は短くなり，t_s/t が大きくなって並列化の効果は減少する.

(2.2) は $n \cdot t = T$ (一定) とすると

$$t=\sqrt{T \cdot t_s},\ n=\sqrt{T/t_s} \qquad (2.4)$$

で最大値をとる．したがって，同期手段によって t_s が決まっている場合に，T 時間かかる処理を $\sqrt{T/t_s}$ 個以上に並列分解することには意味がない.

このように並列処理における同期問題は相対的な問題であり，並列化の効率は t_s/t に依存する.

このことは次のことを意味する.

(1) t_s は小さいことが望ましい.

t_s が小さければ t の小さなオペレーションについても並列化が可能であり，したがって並列処理の多重度を大きくすることができる．たとえば，プログラムBのFがプログラムAのような計算を行う関数の引用であった場合，同期手段が高速であれば図1(b)の④をすべて同図(a)で置き換えることが可能になり，さらに並列度を上げることができる.

(2) t は大きいことが望ましい.

t_s が大きくても t が十分大きければ効率のよい高多重並列処理が可能である．このことは，プログラムを再構成して同期の回数を減少させることによって実現できる．たとえばプログラムBにおいて $N=100$ とし，プロセッサ10台で並列処理を行う場合は**図3**(a)および(b)のような二つの並列プログラムが考えられる．ただし㉖は通常の DO 文，㉗は図1(b)に示した形で制御変数に値を与えて並列実行を開始させる並列 DO 操作する.

これらのプログラムはオペレーションを適当にまとめて P-片とし，(a′) および (b′) のように書くことができる．(b′) においては並列実行されるオペレーションの実行時間が (a′) に比較して非常に大きくなっており，その分だけ同期のオーバヘッドが少なくなっている.

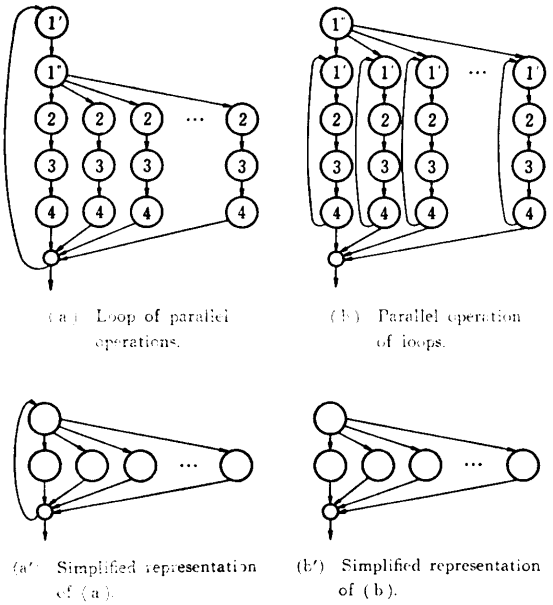


図 3 同期回数の減少
Fig. 3 Diminution of synchronization.

3. 同期操作の高速化

前章の議論で，並列処理の多重度を上げ効率のよい並列処理を行うためには，同期に要する時間 t_s を小さくしなければならないことが明らかになった．この章では FIFO を用いた高速な同期手段を提案する.

3.1 待ちなし並列プログラム

グラフで表現された並列プログラムのあるクラスは，各プロセッサがそのプロセッサ上のオペレーションを先着順 (first come first served, 以後 FCFS と書く) に実行すると自然に同期がとれる.

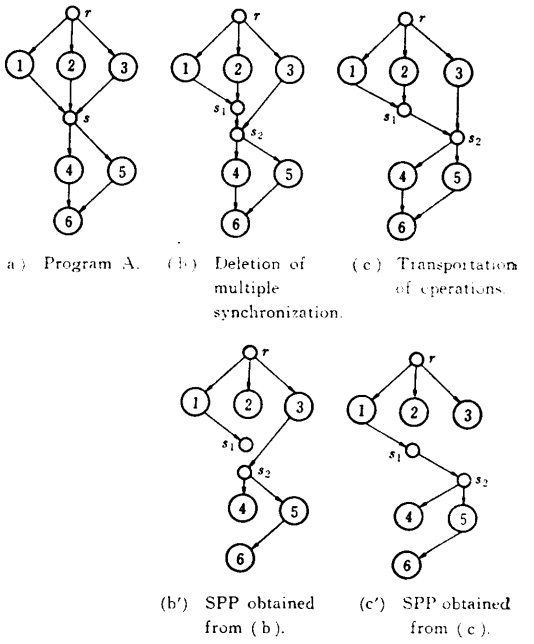


図 4 待ちなし並列プログラム
Fig. 4 Self synchronizing parallel program.

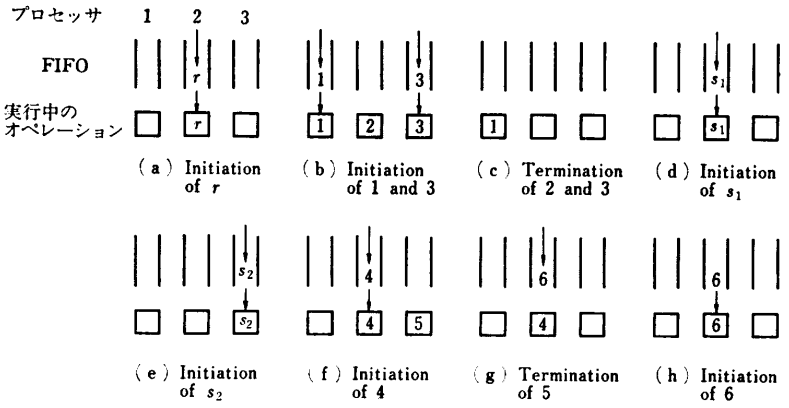


図 5 図 4 (c′) のプログラムの実行例 (①>③>②, ④>⑤ の場合)
Fig. 5 An example of the execution steps of the program (c′) in Fig. 4. (the case of ①>③>②, ④>⑤ as execution time)

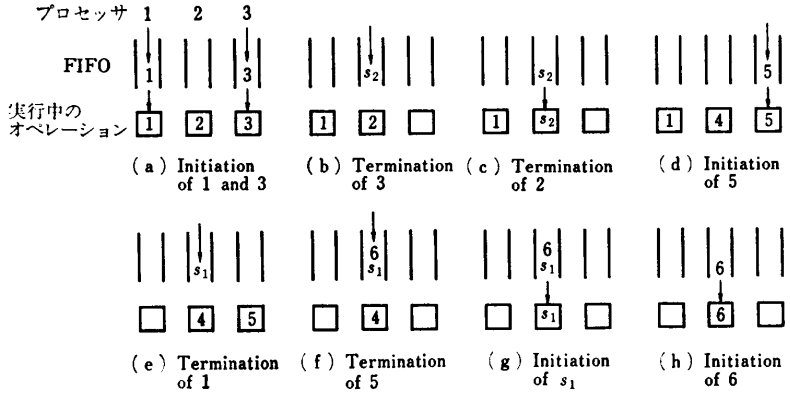


図 6 図 4 (b′) のプログラムの実行例 (①>②>③, ④>⑤ の場合)
Fig. 6 An example of the execution steps of the program (b′) in Fig. 4. (the case of ①>②>③, ④>⑤ as execution time)

たとえば，プログラムAは**図4**(a)のように表現されるが，これを同図(b)，(c)のように変換することができる．

図4(c)は二つのオペレーション間の同期しか含まず，しかも同期操作を実行するプロセッサが(b)とは異なっているが，これらの実行結果はすべて等しい．

図4(c)から同期操作に入る下向きの枝をとり除くと(c′)を得る．各プロセッサがFCFSでオペレーションを実行しているとすると，(c′)の実行結果は(a)の実行結果と等しい．

なぜならFCFSを仮定すると(c)と(c′)はまったく同じ意味をもつからである．実際，(c′)においては①，②が同時に起動されるから，①が完了した後に起動される s_1 は当然②の実行が完了した後に実行される．すなわち，もし s_1 が起動されたとき②がまだ完了していなければ，FCFSの仮定より s_1 の起動要求は実行キューのなかで待たされ，②の完了後に s_1 の実行が開始される．s_2 および⑥についても同様で，これらは前に起動されているオペレーション③および④が完了した後にしか実行されない．プログラム(c′)の実行の状況を**図5**に示す．この場合には(g)で待合せが起こっている．

図4(c′)は同期操作に対応する合流点をもっていない．このように木構造グラフで表現される並列プログラムを待ちなし並列プログラム(self synchronizing parallel program, SPP)と呼ぶ．

図4(c′)はFCFSの仮定のもとで(a)と同じ結果を与えた．しかし(b)から同様の操作で得られる(b′)は，FCFSの仮定のもとでも(a)と同じ結果を与えるとはかぎらない．なぜなら，s_1 と s_2 の実行順序はオペレーション①と③との実行時間の長さによって変わり，①の実行時間が③の実行時間より長いと，s_2 した

がって④，⑤は①が完了する前に実行されてしまうからである．この例を**図6**に示す．(d)において①の完了前に④，⑤が起動されている．

図4 (c′)のように，各オペレーションの実行時間の大小にかかわらず結果が定まる SPP を確定的 SPP，(b′)のように場合によって結果が異なる可能性のある SPP を非確定的 SPP と呼ぶ．SPP は確定的でないと並列プログラムとしての意味をもたない．

一般の並列プログラムがどのようにして，またどのような確定的 SPP に変換できるかは，並列プログラムの機械的変換を考えるとき，重要かつ興味ある問題である．しかしここではこの問題についてはふれない．ここでは以後，確定的 SPP が与えられている，あるいは確定的 SPP を作ることができるとして考える．

3.2 SPP の実行機構

SPP は同期操作をもっておらず，同期は FCFS という逐次操作に還元されている．

SPP において，'同一プロセッサ上にあって一度制御が渡ると中断されずに実行されるオペレーションの集合をタスクと呼ぶ．たとえば図4 (c′)では，{①}，{②}，{s_1} 等はタスクであり，{s_2, ⑤}もタスクである．

SPP の実行単位はタスクであり，FCFS による SPP の実行管理機構は以下のように構成される．

各プロセッサはそのプロセッサ上のタスクの実行順序を制御するため，タスクの入口番地を保持する FIFO キューと，その上の次の二つの操作をもつ．

(1) タスク切換

一つのタスクの実行が完了すると，プロセッサは自分のキューから次に実行すべきタスクを取り出し，その入口番地に制御を渡す．このときキューが空であればプロセッサはアイドル状態となり，次に実行すべきタスクがキューに登録されるのを待つ．

(2) 並列分岐

他のプロセッサに割り当てられたタスクの起動は，そのタスクが割り当てられているプロセッサのキューに，そのタスクの入口番地を登録することによって実現される．タスクを起動した側のプロセッサは引続き実行を続けることができるので，この操作は並列分岐となる．

図4 (c′)において，異なるプロセッサ上のオペレーションに入る枝は並列分岐命令の実行を意味し，同一プロセッサ上の他のオペレーションに入る枝をもたないオペレーション，たとえば①，②，③，s_1 等はそのオペレーションの最後でタスク切換命令が実行されることを意味する．

このように SPP では，並列実行を意味する制御の移動と同期を意味する制御の移動とが，同じ並列分岐命令で実現される．

FIFO キューは FIFO メモリとしてハードウェアで簡単に実現でき，タスク切換命令および並列分岐命令はいずれも数バスオペレーションの機械語命令として構成することが可能である．したがってこのような機械語命令をもつシステムにおいては，並列実行される SPP の各オペレーションをかなり小さくでき，2章で述べたように効率のよい柔軟な高多重並列処理が可能となろう．

3.3 タスク管理ハードウェア

ここで対象とするシステムは，**図7**に示すようなメモリ共有型のモジュラ型多重プロセッサシステムである．pm_i はプロセッサモジュールで，プロセッサユニット pu_i とメモリユニット mu_i とから成り，アクセス機構によって結合されている．ここに i はモジュール識別番号で，これをシステムアドレスと呼ぶ．

アドレス空間はシステムアドレスとロケーションから成る二次元アドレスで，各 pu は任意の mu の内容を直接参照することができる．図7からわかるように，mu_i は pu_i から内部的に参照されると同時に，アクセス機構を通じて他の pu からも参照される．すなわち mu は，ローカルメモリであると同時にグローバルメモリでもある．

mu_i に置かれたタスクは pu_i によって実行される．したがって命令フェッチに関してはメモリアクセスの競合は起こらない．

pu は，load, store, add, compare, branch 等の通常の機械語命令体系をもつ．ただし，分岐命令を除くメモリ参照形の命令では，実効アドレスがシステムアドレスの分だけ拡張されている．分岐命令はそのモジュール内での相対的分岐である．

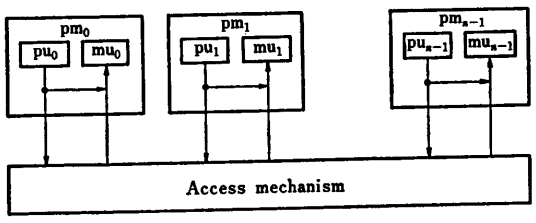


図7 システムモデル
Fig. 7 System model.

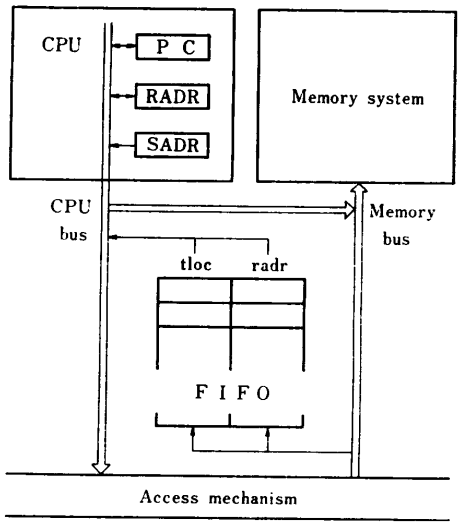


図 8 タスク管理ハードウェア
Fig. 8 Hardware configuration of the task manager.

pu はさらに，タスク管理のための命令，parallel branch および exchange task をもつ．タスク管理に関係するハードウェアを**図 8** に示す．各要素は次のような意味をもつ．

FIFO：タスクの実行順序を制御する FIFO メモリ．tloc は起動するタスクの入口のロケーションを示し，radr はこのタスクを起動した parallel branch 命令の次のアドレスを示す．

RADR：実行中のタスクの radr を保持しているレジスタ．radr は通常の branch and link 命令における戻り先アドレスと同様，パラメータの受渡し等に使用される．

SADR：このモジュールのシステムアドレスを保持しているレジスタ．

PC：プログラムカウンタ

並列分岐命令およびタスク切換命令は，このハードウェアの上で以下のように動作する．

（1） parallel branch *address*

address はシステムアドレス dsad とロケーション loc から成る．この命令は dsad で指定されたモジュールの FIFO の tloc 部に loc を，radr 部に SADR および PC の内容を書き込む．

（2） exchange task

FIFO を読み出し，radr 部を RADR に，tloc 部を PC に格納する．この命令の実行によりこのプロセッサの制御は新しいタスクに移る．なお FIFO が空であればプロセッサはアイドルとなり，他のプロセッサがこのモジュール上のタスクに対して parallel branch を出すまでウェイト状態になる．

FIFO をメモリ空間中のレジスタとして構成すれば，parallel branch および exchange task 命令はそれぞれ store 命令，load 命令の 2 倍程度の実行時間で実現できよう．

4. 応 用 例

この章では FIFO による同期手段が有用であることを示すため幾つかの例を示す．ハードウェアのモデルは，3.3 節に示した並列分岐命令およびタスク切換命令をもった，メモリ共有型のマルチプロセッサシステムとする．なお簡単のため，機械語命令の実行時間はすべて同一時間 t であるとし，並列に実行されるオペレーションは同一実行時間をもつとする．

4.1 セマフォ型同期

図 2（b）に示した n 個のオペレーションの同期にはカウンタを使用したセマフォ型の同期が使用される．

このシステムの上では，セマフォ型の同期は次のようにして実現できる．

```
        プログラム  C
SYNC : load r, COUNT
       decrement r, 1
       store r, COUNT
       branch on zero PROC
       exchange task
PROC : ⋮
```

COUNT を N に初期設定しておけば，N 個のタスクが SYNC に対して parallel branch を実行すると PROC からの処理が始まる．1 個の同期に parallel branch 1 回と SYNC 1 回の実行が必要なので，N 個の同期に必要な時間 t_{s1} は次で与えられる．

$$t_{s1} \simeq 6\,Nt \qquad (4.1)$$

wait-post など割込みを用いた同期操作では，レジスタの退避や割込み解析，パラメータの受け取り，制御表の参照・更新，実行待ち行列の操作等の処理が必要であり，オペレーティングシステムの規模・構成法によっても異なるが，1 個の同期マクロの実行に $100\,t$ 程度の実行時間が必要となろう．一方 SYNC は exchange task 命令によって起動されるので，これに比較すると格段に高速である．

図 2（b）は**図 9**（a）のような SPP で表現できる．この場合は，1 個の同期に exchange task 命令が 1 回，parallel branch 命令が 1 回実行されるので，N 個の同期に要する時間 t_{s2} は次のようになり，さらに

高速な同期手段となっている.

$$t_{s2} \simeq 2Nt \tag{4.2}$$

以上は集中形の同期であるが，同期回数 N が大きいときは分散同期が有効である．図9(b)の形の分散同期では同期段数が $\log_2 N$ となり，1段の同期に $10t$ 時間かかるので同期時間 t_{s3} は次のようになる.

$$t_{s3} \simeq (10 \cdot \log_2 N)t \tag{4.3}$$

さらにこれを図9(c)のように変換すると同期に要する時間 t_{s4} は

$$t_{s4} = [2m + 10 \cdot \log_2(N/m)]t \tag{4.4}$$

となり，$m=8$ で同期時間は最小となる．このとき，

$$t_s \simeq (10 \cdot \log_2 N - 14)t \tag{4.5}$$

である.

なお，セマフォ型同期を使用すると並列プログラムはもはや木構造にはならない．このような並列プログラムは，木構造をもった SPP を並列分岐および合流のオペレーションで結合した形になっている.

4.2 相互排除

相互排除も同期の一種であり，SPP で簡単に実現できる．複数のプロセスが並行して動作することを禁止するクリティカルセクションは，**図10**(a)に示すようにその部分を同一プロセッサに割り当てるだけでよい．また，モニタモジュールなどのような排他的処理を含んだ再入可能なプログラムは図10(b)のように構成すればよい．プログラムSが共用ルーチンであり，プログラムPおよびプログラムQから parallel branch で呼ばれているとする．SはPに対する①の処理が完了するとPに対する②の処理を起動し，exchange task によってQに対する①の処理を開始する.

これは一種のパイプライン処理である.

4.3 行列演算

行列演算は SIMD (単命令多データ) 型処理が効率がよいとされているが，同期手段が高速であるならMIMD 型マルチプロセッサでも十分効率よく処理できる．ここでは行列の和および積の計算を考える.

4.3.1 行列の和

行列の和を求めるプログラムは**表1**(a)のように書かれる．**図11**に示すように各行列が N 台のプロセッサに分散配置されているとすると，内側のループは各プロセッサごとに独立に，アクセス競合なしに実行できる．このとき各プロセッサ上のプログラムは表1(b)のように書け，この場合は N 個のタスクを起動するために $\log_2 N$ 回の parallel branch 命令の実行が必要である．また，図9(c)の形の同期をとると

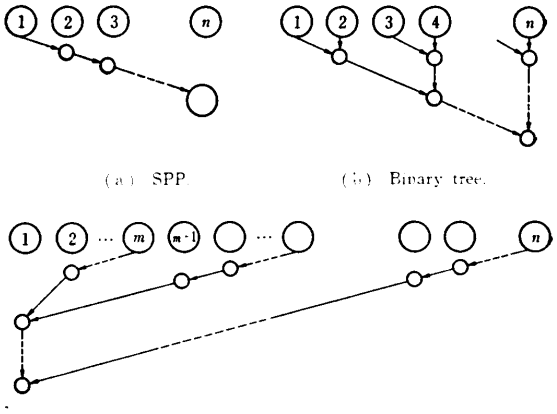


図 9 同期の方式
Fig. 9 Synchronization scheme.

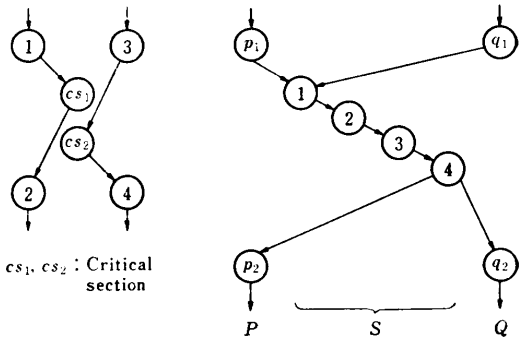


図 10 相互排除
Fig. 10 Mutual exclusion.

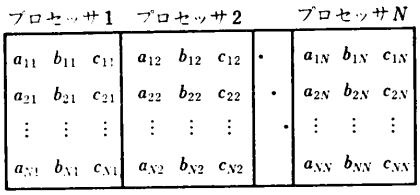


図 11 行列の分散配置
Fig. 11 Arrangement of matrix elements.

すると同期に要する時間は (4.5) で表されるので，全体の実行時間は $(10N+11 \cdot \log_2 N - 13)t$ となる.

逐次実行の実行時間を t_{seq}，並列実行の実行時間を t_{par} とし，実効多重度を

$$n = t_{seq}/t_{par} \tag{4.6}$$

で定義すると，$N=256$ の場合は $n=274.3$ となり，十分高い実効多重度を得ることができる.

4.3.2 行列の積

行列の積を求めるプログラムは表1(c)のように書

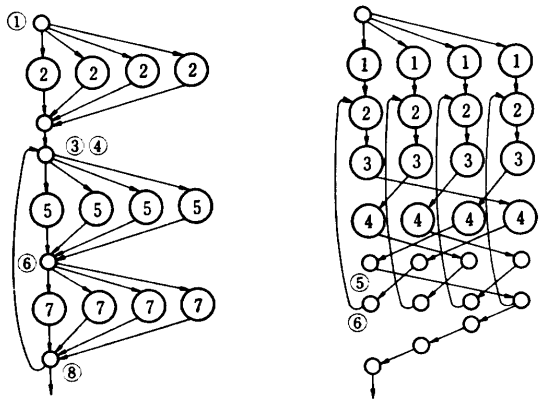
表 1 プログラム例と実行ステップ数
(PDP 11/24 FORTRAN IV-PLUS)
Table 1 Execution steps of example programs.

	プログラム	実行ステップ数
(a)	10 DO 11 $J=1,N$ DO 11 $I=1,N$ 11 $C(I,J)=A(I,J)+B(I,J)$	$(11N+7)N+4$
(b)	20 DO 21 $I=1,N$ 21 $CJ(I)=AJ(I)+BJ(I)$	$10N+1$
(c)	30 DO 31 $J=1,N$ DO 32 $I=1,N$ $S=0.0$ DO 33 $K=1,N$ 33 $S=S+A(I,K)*B(K,J)$ 32 $C(I,J)=S$ 31 CONTINUE	$((16N+11)N+7)N+2$
(d)	① 40 DO 41 $J=1,N$ ② DO 41 $I=1,N$ 41 $C(I,J)=0.0$ ③ DO 44 $K=1,N$ ④ DO 42 $J=1,N$ ⑤ $L=\mathrm{MOD}(K+J-2,N)+1$ $BL=B(L,J)$ DO 42 $I=1,N$ $C(I,J)=C(I,J)+A(I,J)*BL$ 42 $W(I,J)=A(I,J)$ ⑥ DO 43 $J=1,N$ ⑦ $M=\mathrm{MOD}(J,N)+1$ DO 43 $I=1,N$ 43 $A(I,J)=W(I,M)$ ⑧ 44 CONTINUE	$((28N+36)N+10)N+7$
(e)	① 50 DO 51 $I=1,N$ 51 $CJ(I)=0.0$ ② DO 52 $K=1,N$ $L=\mathrm{MOD}(K+J-2,N)+1$ $BJL=BJ(L)$ ③ DO 53 $I=1,N$ $WJ(I)=AJ(I)$ 53 $CJ(I)=CJ(I)+WJ(I)*BJL$ GO TO $(J-1,54)$ ④ 54 DO 55 $I=1,N$ 55 $AJ(I)=WJ1(I)$ GO TO $(J+2,56)$ ⑤ 56 GO TO $(J-1,52)$ ⑥ 52 CONTINUE	$(20N+31)N+6$ GO TO (j,s) は含まない.

かれる．前例と同様に各行列が配置されているとき，このプログラムは J について並列実行が可能である．しかしこの場は前例とは異なってデータアクセスが全メモリに広がっており，N が大きい場合にはデータアクセスの距離およびアクセス競合が大きくなる．このプログラムは計算順序を変えて表 1 (d)のように書くことができる．これより**図 12**(a) のような並列プログラムが得られる．

表 1 (d)において配列 A と W とのデータの依存関係は，J と $M(=J+1)$ との間にだけ存在するのであるから，K のループを各プロセッサに配分すると図 12 (b)のような待ちなし並列プログラムが得られる．このときプロセッサ J 上のプログラムは表 1 (e)のように書ける．ここで AJ, WJ 等はモジュール J に，$WJ1$ はモジュール $J+1$ に割り当てられた一次元の配列であり，GO TO (j, s) はモジュール j 上の文番号 s の文に並列分岐した後タスク切換を行う命令とする．

このプログラムではデータのアクセスは局所化されていて，すべてのプロセッサは隣りのモジュールのメ



(a) Parallel program. (b) SPP.

図 12 プログラム(d)から得られる並列プログラム
Fig. 12 Parallel programs obtained from program (d).

モリしか参照せず，したがって同期も分散同期になっている．GO TO(j, s) が parallel branch, exchange task の2命令で実行されるとし，前例と同様の同期方法をとるとすると，このプログラムの実行時間は $((20N+37)N+11 \cdot \log_2 N-8)t$ となる．

$N=256$ とすると表1(c)に対して $n=204.1$ となり，かなり高い実効多重度を得ることができる．

なおこの例では積和という比較的簡単な演算を対象としたため，データの移動および同期の時間の影響が大きい．並列化の対象となる演算が複雑であれば実効多重度はさらに向上すると考えられる．たとえば表1(e)の元になった表1(d)のプログラムを対象にすると，$n=357.6$ となり非常に高い実効多重度を得る．ここで $n>N$ となっているのは配列の次元が下がるため番地計算が少なくなるためである．

5. むすび

以上 MIMD 型並列処理システムにおける同期問題について考察し，高速な同期手段として FIFO キューを用いる待ちなし並列プログラム (SPP) の概念とその実行管理ハードウェア，および幾つかの例題を示した．

SPP はデータ依存関係に基づいて構成された一種のデータ駆動型プログラムとみなすことができる．しかし SPP はオペレーションとその間の制御フローで表現されているので，通常のいわゆるノイマン型計算機の複合体で実行可能であり，ノイマン型計算機によるデータフロー計算機の一つの実現と考えることができよう．

SPP に関しては，実行管理ハードウェアの実装設計，逐次的プログラムから SPP への変換手順，効率のよいアクセス機構の実現等の問題がある．これらについては別稿で報告したい．

謝 辞 本論文をまとめるに当たり九州大学工学部牛島和夫教授，宇津宮孝一助教授および荒木啓二郎助手には種々のご助言をいただいた．ここに記して謝意を表す．

参考文献

1) Flynn, M. J.: Very High Speed Computing Systems, *Proc. IEEE*, Vol. 54, pp. 1901-1905 (1966).
2) 関根 陽：分散処理技術，情報処理，Vol. 20, No. 4, pp. 275-283(1979).
3) Bear, J. L. : A Survey of Some Theoretical Aspects of Multiprocessing, *Comput. Surv.*, Vol. 5, No. 1, pp. 31-80(1973).
4) Maekawa, M.: A Classification of Process Coordination Schemes in Descriptive Power, *Int. J. Comput. Inf. Sci.*, Vol. 9, No. 5, pp. 383-406(1980).
5) 有田五次郎：並列プログラムの一形式化とその変換について，信学技報，EC 77-58, pp. 25-32 (1978).
6) 有田五次郎：FIFO キューを同期手段とする待なし並列プログラムとその実行管理機構について，情報処理学会第23回プログラミングシンポジウム報告集 (1982).

(昭和57年6月7日受付)
(昭和57年10月4日採録)