多段結合ネットワークを用いる超並列マシンのためのパイプライン化 MIMD プロセッサ†

森下 巖††

本論文は，多段結合ネットワークを用いる共有メモリ型超並列マシンに使用する目的のプロセッサのアーキテクチャについて検討したものである．このタイプの超並列マシンの実現においては，多段結合ネットワークの大きな伝送遅延と，単位時間に伝送可能な要求数の制限が大きな問題になる．経済性を考慮に入れると，プロセッサ要素の動作速度と，多段結合ネットワークおよびメモリ要素の動作速度の間にある程度のギャップを許容しなければならない．本論文においては，多段結合ネットワークとメモリ要素で構成される主メモリについて，伝送遅延ネックとスループット・ネックを区別して検討し，キャッシュメモリ単独ではスループット・ネックは解消できるが伝送遅延ネックの解消は困難であることを示し，パイプライン化 MIMD プロセッサの採用を提案する．提案したアーキテクチャのプロセッサは，多段結合ネットワークとメモリ要素で構成される主メモリがもともとパイプライン化されている事実を利用し，多数の命令流をパイプライン実行することによって，主メモリの伝送遅延ネックを本質的に解消しようとするものである．スループット・ネックはキャッシュメモリの併用によって解消させる．最後に，アーキテクチャの具体例を提示し，必要となる汎用レジスタの総数を検討し，1チップ集積の可能性について議論する．

1. はじめに

MIMD 超並列マシンのハードウェア・アーキテクチャは，多数のコンピュータ要素をメッセージ通信路で結合したメッセージ通信型と，多数のプロセッサ要素と多数のメモリ要素を結合ネットワークで結合した共有メモリ型に分類できる．両者のなかでは共有メモリ型がプログラミングの自由度がより大きいと思われる．ハードウェア上に実現する並列計算パラダイムとしてもメッセージ通信方式と共有メモリ方式が考えられるが，共有メモリ型アーキテクチャを用意すると，共有メモリ方式はもちろんのこと，メッセージ通信方式も効率よく実現できるからである．

純粋な共有メモリ型アーキテクチャの一つの利点は，多数の命令流の並列実行において，あらかじめ各命令流を特定のプロセッサ要素に割り当てる必要がないことである．ある命令流がサスペンドしたとき，どのプロセッサ要素でも実行を再開始することができる．すべての命令流の内部状態を単一の共有メモリ上に格納するので，プロセッサ要素も一つの資源として取り扱うことが可能となる．

共有メモリ型アーキテクチャの結合ネットワークとして，多段結合ネットワークが提案されている．プロセッサ要素の個数を N 個とするとき，クロスバスイッチでは $N \cdot N$ 個のスイッチ要素が必要になるのに対して，多段ネットワークではもっとも単純な 2×2 のスイッチ要素を用いる場合でも $N\log_2 N$ 個しか必要でない．最近，多段結合ネットワークを用いる共有メモリ型超並列マシンの研究が盛んになっている[1)~3)]．

しかし，多段結合ネットワークを用いる場合には，メモリアクセス要求の発信から受信までに大きな伝送遅延が発生し，また，単位時間に伝送できる要求量にも制限がある．しかも，経済性を考慮に入れると，プロセッサ要素の動作速度と，多段結合ネットワークおよびメモリ要素の動作速度の間にはある程度のギャップを許容するのが望ましい．実際，プロセッサ・チップのクロック周波数は上昇を続けており，とくに RISC 型とすると高いクロック周波数の実現が容易であって，50MHz のクロック周波数もすでに現実のものとなっている．しかし，超並列マシンで必要となる非常に大規模な多段結合ネットワークを 50 MHz のクロックで動作させるのは簡単ではない．経済性を考慮に入れると，多段結合ネットワークはプロセッサ要素のクロック周波数よりも低い周波数で動作させるのが望ましい．同様に，メモリ要素用の大容量メモリ・チップも，ある程度は低動作速度のものを使用せざるをえない．

問題は，多段結合ネットワークおよびメモリ要素と

† A New Pipelined MIMD Processor for Large Scale Parallel Machines with Multistage Interconnection Networks by IWAO MORISHITA (Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科

して低動作速度のものを使用した場合でも，マシン全体の処理速度を低下させない手段が存在するかどうかである．本論文では，プロセッサ要素にキャッシュメモリを格納する方式では伝送遅延の問題を解決するのは困難であることを示し，パイプライン化 MIMD プロセッサとキャッシュメモリを併用する方式を提案する．

パイプライン化 MIMD プロセッサの基本原理は新しいものではない．M. J. Flynn により "Shared Resource MIMD" として提案されたものである[4]．この方式の具体的な実現としては Denelcor 社のマルチプロセッサ・マシン HEP があり[5),6)]，そこではパイプライン実行の側面を強調して "Pipelined MIMD" と呼んだ．HEP では主メモリはパイプラインの外にあったが，後藤英一ほかは，主メモリをもパイプライン化したアーキテクチャ (cyclic pipeline architecture) を考え，FLATS 2 と呼ぶ命令流2個の単一プロセッサ・マシンの開発を進めている[7),8)]．ここでは，多段結合ネットワークを用いる共有メモリ型超並列マシンにおいては，主メモリがもともとパイプライン化されている事実を利用し，本論文が目的とする用途に適したパイプライン化 MIMD プロセッサを提案する．

2. 多段結合ネットワークを用いる場合の問題点

図1に示す構成の実現を考える．経済性を考慮に入れ，プロセッサ要素の動作速度と，多段結合ネットワークおよびメモリ要素の動作速度にはギャップがあると仮定する．プロセッサ要素と多段結合ネットワークとの速度ギャップを s_N，メモリ要素との速度ギャップを s_M とし，

プロセッサ数 : N
ネットワーク段数 : $n=\log_2 N$
プロセッサ・クロック周波数 : f_P
ネットワーク・クロック周波数 : $f_N=f_P/s_N$
メモリ要素単体のアクセス時間 : s_M

であるとする．時間はすべてのプロセッサ・クロック単位で表現する．$s_N, s_M=1$ はギャップのない場合である．実際問題としては，$s_N, s_M=2$〜3 程度を考えればよい．

プロセッサ要素側から見れば，多段結合ネットワークに結合されたメモリ要素が主メモリとして機能する．この主メモリについては，下記の2種類のネックを検討する必要がある．

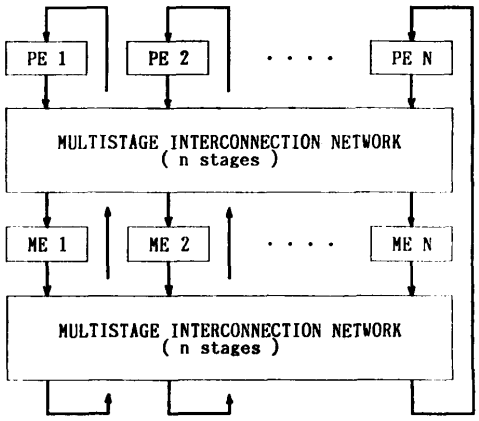

図 1 多段結合ネットワークを用いる共有メモリ型超並列マシンの基本構成
PE: プロセッサ要素，ME: メモリ要素
Fig. 1 Block diagram of a shared memory parallel machine with one pair of multistage interconnection networks.
PE: processor element, ME: memory element.

（1） 伝送遅延ネック

メモリアクセスに大きな遅延が発生する．メモリ要素に読み出し要求を発信してからデータを受信するまでの遅延時間の最小値 D_{min} は，最小伝送遅延時間とメモリアクセス時間の和であり，

$$D_{min}=2ns_N+s_M \quad (1)$$

となる．プロセッサが命令フェッチの要求を発信してから（プログラム・カウンタ PC の指定する番地の内容の読み出し要求を発信してから），命令がプロセッサに到着するまでに少なくとも D_{min} クロックが必要である．これはデータ読み出しの場合も同様である．実際にはこれにネットワーク内の要求の衝突による待ち時間が加わる．この間，プロセッサがアイドリングすると処理速度は大幅に低下する．

（2） スループット・ネック

ネットワークの1プロセッサあたりのスループットは，1プロセッサ・クロックごとにネットワークが伝送できる要求数と定義する．多段結合ネットワークの最大スループット RN_{max} はネットワークの構成法に依存するが，構成を工夫すると1要求/1ネットワーク・クロックが実現できる[9)]．したがって，ネットワークの最大スループットは，

$$RN_{max}=1/s_N$$

（要求数/プロセッサ・クロック）

となる．たとえば，$s_N=2$ の場合，最大スループット

は2クロックごとに1個の要求発信，返信受信に低下する．実際には，この最大スループット RN_{max} で要求を伝送しようとすると遅延時間が急激に大きくなるので，動作時のネットワークに対する負荷はこれよりある程度低くとる必要がある．すなわち，$s_N=1$ とする場合でも，1プロセッサ・クロックごとに1個の要求を発信することはできない．

一方，メモリ要素が処理できる要求数は，

$$RM_{max}=1/s_M$$

(要求数/プロセッサ・クロック)

である．したがって，主メモリ全体の最大スループット R_{max} は，

$$R_{max}=1/\max(s_N, s_M) \quad (2)$$

となる．結局，プロセッサが1クロックごとに1メモリアクセスを必要とする場合にはスループット・ネックが発生することになる．

3. キャッシュメモリの効果

上記のネックに対する対策としては，プロセッサ要素内にキャッシュメモリを格納する方式が考えられる．キャッシュメモリは1クロックでのアクセスが可能であり，キャッシュがヒットすれば，伝送遅延ネックの問題もスループット・ネックの問題も同時に解決される，というアイデアである．

たしかに，単一プロセッサ・マシンの場合（プロセッサと低速度主メモリが直結されている場合）には，プロセッサ内にキャッシュメモリを格納する方式は有効である．数 % のキャッシュミスがあっても，処理速度はほとんど低下しない．問題は，多段ネットワークを用いる場合でもキャッシュメモリが十分に有効性であるかどうかである．

3.1 伝送遅延ネック

議論を簡単にするため，プロセッサは RISC 型であり，このプロセッサでは，ある命令を実行するのに必要なクロック数は，命令フェッチから実行完了までに必要なメモリアクセス回数に等しいとする．たとえば，レジスタ間の演算命令は1クロック，1語に対する LOAD, STORE 命令は2クロックである．すなわち，命令フェッチをも含めて k 個のメモリアクセスの必要な命令の実行時間は k であり，この命令を k 型と呼ぶことにする．

いま，k 型の命令の発生する確率を p_k とすると，1命令の実行に必要な平均クロック数 m は，1命令の実行に必要な平均メモリアクセス回数に等しく

$$m=\sum_k p_k k \quad (3)$$

で与えられる．通常のプログラムでは m は1より数十 % 程度大きな値になると言われている．

いま，多段ネットワークを用いる場合のミス率を a とする．ミスが発生すると，すくなくとも $2ns_N+s_M$ のアイドリングが発生するので，ミス率が a であるときの1命令の実行に必要な平均クロック数 $r(a)$ は

$$r(a)\geq m+(2ns_N+s_M)ma \quad (4)$$

となる．一方，単一プロセッサの場合（プロセッサと主メモリが直結されている場合）のミス率を a_1 とすると，1命令の実行に必要な平均クロック数 $r_1(a_1)$ は，

$$r_1(a_1)=m+s_M ma_1 \quad (5)$$

となる．したがって，多段ネットワークを用いる場合に単一プロセッサの場合と同一の処理速度を得るためには，

$$m+(2ns_N+s_M)ma\leq m+s_M ma_1$$

が成立しなければならない．すなわち，

$$\frac{a}{a_1}\leq\frac{1}{2n(s_N/s_M)+1} \quad (6)$$

となる．多段ネットワークを用いる場合には，**表1**に示すように，ミス率 a を非常に小さい値に保つ必要がある．

逆に言えば，多段ネットワークを用いる場合，ミス率 a を単一プロセッサの場合と同一，すなわち，

$$a=a_1$$

とすると，1命令を実行するのに必要な平均クロック数は，

$$m+s_M ma$$

から

$$m+(2ns_N+s_M)ma$$

に増加する．すなわち，単位時間あたりの命令実行数

表1 単一プロセッサ・マシンと同一命令実行速度を実現するためのキャッシュメモリのミス率 a の上限値 ($s_N=s_M$ の場合)

Table 1 Upper limit of the cache memory miss rate for obtaining the same execution speed as in a single-processor machine.

N	n	a_1	
		0.05	0.02
256	8	0.0029	0.00176
1,024	10	0.0024	0.00095
4K	12	0.0020	0.00080
64K	16	0.0015	0.00061

a_1: 単一プロセッサ・マシンのミス率.

表 2 ミス率が単一プロセッサ・マシンと同一である場合の命令実行速度の低下率 $R(a)$
Table 2 Execution speed down when the cache miss rate is the same as in a single-processor machine.

(a) $s_N=s_M=2$

N	n	a	
		0.05	0.02
256	8	0.41	0.62
1,024	10	0.35	0.57
4K	12	0.31	0.52
64K	16	0.26	0.45

(b) $s_N=s_M=3$

N	n	a	
		0.05	0.02
256	8	0.32	0.52
1,024	10	0.28	0.47
4K	12	0.24	0.42
64K	16	0.19	0.36

の減少率 $R(a)$ は,

$$R(a)\leq\frac{1+s_Ma}{1+(2ns_N+s_M)a} \qquad (7)$$

となり，**表 2** に示すように，大幅な処理速度の低下が発生する．

3.2 スループット・ネック

一方，キャッシュメモリは主メモリのスループット・ネックを解消するのには有効である．キャッシュの内容を繰り返し使用することによって，メモリアクセスの回数を減少させることができるからである．

ライトバック方式を採用すると，主メモリアクセスが必要になるのはミス発生時のみであり，1命令あたりのミスの平均発生数は ma である．

p_D：キャッシュ内容が書きかえられている確率

とすると，メモリ要素へ書き戻しも含めた平均要求発信数は

$(1+p_D)ma$

となる．一方，主メモリの最大スループットは1命令あたり $m/\max(s_N, s_M)$ であるので，負荷率を f_L とすると，ネットワークが受け付けることのできる1命令あたりの要求数は

$f_Lm/\max(s_N, s_M)$

となる．したがって，要求が滞留しないためには，

$f_Lm/\max(s_N, s_M)\geq(1+p_D)ma$

すなわち，

$$a\leq f_L/(1+p_D)\max(s_N, s_M) \qquad (8)$$

表 3 スループット・ネックを解消するためのキャッシュメモリのミス率 a の上限値（ただし，負荷率 $f_L=0.75$, $s_N=s_M$ の場合）
Table 3 Upper limit of the cache memory miss rate for working within the memory throughput.

p_D	s_N	
	2	3
0.2	0.31	0.21
0.3	0.29	0.19
0.4	0.27	0.18

の条件が満足されればよい．負荷率 f_L が 75% である場合のミス率の上限を**表 3** に示す．これは容易に実現できる値である．逆に言えば，ミス率を表3の値に止めることができれば，ネットワークの負荷率は75%となる．

以上の検討より，キャッシュメモリの格納によって，スループット・ネックは解消できるが，伝送遅延ネックの解消は困難であることが知れる．

4. パイプライン化 MIMD プロセッサ

キャッシュメモリの格納だけでは，多段ネットワークとメモリ要素に起因する伝送遅延ネックの問題を本質的に解決することはできない．以下では，プロセッサとして，パイプライン化 MIMD，すなわち，多数の命令流をパイプライン実行するアーキテクチャを採用することを提案する．

超並列マシンでは，当然のこととして，十分に多数の並列度が依存する応用プログラムを対象として考えている．提案する方式では，

- 対象の並列実行度はプロセッサの台数 N よりもはるかに大きい．

すなわち，プログラム実行中の大部分の時間においては，N よりもはるかに多数の命令流を並列に実行できると仮定する．

4.1 動作原理

(1) 基本動作原理

広く採用されているように，個々の命令は複数のステップに分解してパイプライン実行する．パイプラインの段数は原理的には任意であるが，以下の検討から知れるように，パイプラインの全段内には1個のメモリアクセスしか含まない方式とするのが有利である．そこで，RISC 型の方式を模倣し，

- FS：フェッチ発信

- FR：フェッチ受信
- D ：デコードおよびオペランド読み出し
- E ：実行
- W ：結果書き込み

の5ステップとする．

ここで，「q個の命令流をパイプラインのステップ単位で並列に実行させる」ことを考える．各命令流の命令系列を

第1命令流：S1-1, S1-2, S1-3, …
第2命令流：S2-1, S2-2, S2-3, …
…

と書く．いま，メモリアクセスが命令フェッチのみである命令流を考えると，8命令の場合，**図2**(a)のパイプライン実行が可能になる．このパイプラインにおいては，

① 命令フェッチにおいて，5クロック（一般には$q-3$）の遅延を置くことができる．
② 各命令流とも，一つの命令の実行が完了してから次の命令のフェッチを開始できる．

すなわち，qを大きくとればメモリアクセスに任意の遅延を許容できるだけでなく，②の性質により分岐によるパイプラインの乱れも発生しない．遅延分岐などの小細工は不要となる．

多段ネットワークとメモリ要素で構成される主メモリは，本質的にパイプライン化されている．これが，q個の命令流によって共用される．この主メモリMをも含めれば，同図(b)に示す8段（一般にはq段）のパイプライン実行となる．

すでに述べたように，この基本原理は新しいものではない[4]．しかし，**最初の提案では主メモリのパイプライン化は考えなかった．実際，HEP** ではコードメモリはプロセッサに直結し，多段ネットワークに結合したデータメモリはパイプラインの外にある[5),6]．主メモリをもパイプライン化することを提案したのはFLATS 2であるが，対象が単一プロセッサ・マシンであり，単一命令流の処理速度を上げるのが目的であるので，多段ネットワークの使用は考えていない[7),8]．パイプライン段数はできるだけ多く10段にとり，先行制御を採用して命令流は2個に押さえている．

多段結合ネットワークを用いる超並列マシンの場合には，図2から知れるように，主メモリをもパイプライン化したパイプライン化 MIMD がもっとも自然な形で適用できる．この方式を採用してq個の命令流を並列に実行させるためには，

① q個の命令流の内部状態を保持するものとしてq組のプログラム・カウンタ PC，ステイタスレジスタ SR，汎用レジスタセット
② ネットワークの受信バッファとしてq個のレジスタ BR（有効フラグ付属）

を用意すればよい．①は自明である．②が必要になるのは，ネットワークからの返信は必ずしも発信の順序では返送されず，後から発信したものが先に返送される可能性があり，また，所定の時刻に返信が到着していない可能性もあるからである．ある命令流に対して返信が到着するとその命令流用のBR に格納し，有効フラグをセットする．FR のステップにおいては，まず，BR の有効フラグをテストし，有効であれば読み出しを実行し，無効であれば（返信未到着であれば）そのマシンサイクルではアイドリングして次のマシンサイクルまで待つ（1マシンサイクルの損失）．

なお，もともとパイプライン化されているデコーダや ALU は各1個でよい．

（2） LOAD, STORE の2マシンサイクルによる実行

一つの命令流については，**図3**(a)に示すように命令が1個1個

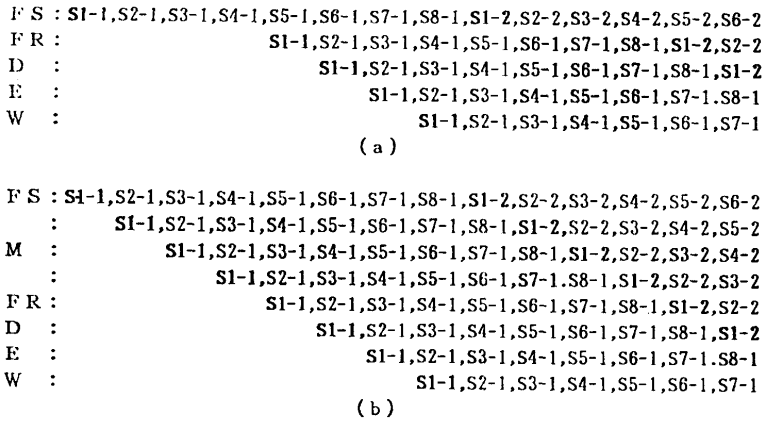

図2 多重命令流のパイプライン実行
FS：フェッチ発信，FR：フェッチ受信，D：デコード，E：実行，W：書き込み，M：主メモリアクセス
Fig. 2 Pipelined execution of multiple instruction streams.
FS: fetch send, FR: fetch receive, D: decode, E: execution, W: write, M: memory access.

順番に実行される．単位となるマシンサイクルはqクロックである．命令流中に LOAD や STORE が出現したとすると，これの実行にはメモリアクセスが必要になるが，同図(b)に示すように，第2マシンサイクルを消費して実行することにすれば，他の命令流の実行には何の影響も与えない．処理速度としては，RISC 型の場合と同様に1クロック増加するだけである．第1マシンサイクルのステップEにおいてアドレス生成を実行し，第2マシンサイクルの命令フェッチの時間をメモリアクセスにあてる．

図4にプロセッサの基本構成とパイプライン実行ステップを示す．プロセッサには出力ポートと入力ポートの2ポートを用意する．

（3） 例外処理の複数マシンサイクルによる実行

ある命令流に，プロセッサ内部に起因するトラップなどの例外処理が発生した場合には，上と同様に，その命令流の複数マシンサイクルを消費して処理する．すなわち，必要な個数のマシンサイクルを消費して PC, SR などをメモリ要素に書き込んだあと，PC に分岐先アドレスを格納して処理を終了する．

外部から起動される割り込みやリセットも，基本的には同様に処理できる．

（4） 浮動小数点ユニットの組み込み

通常の RISC 型プロセッサにおいては，多数のステップ数を必要とする乗除算ユニットを組み込むのは困難である．しかし，上記の方式では乗除算ユニットでも浮動小数点ユニットでも簡単に組み込むことができる．具体的には演算ユニットをパイプライン化し，第2マシンサイクルの命令フェッチの時間に実行させる．**図5**に示すように，パイプライン化した浮動小数点ユニットは，多段ネットワークに結合されたメモリ要素と同様に，q個の命令流が共用する．パイプライン化 MIMD が，最初，“shared resource MIMD” として提案されたのはこの性質を持つからである．

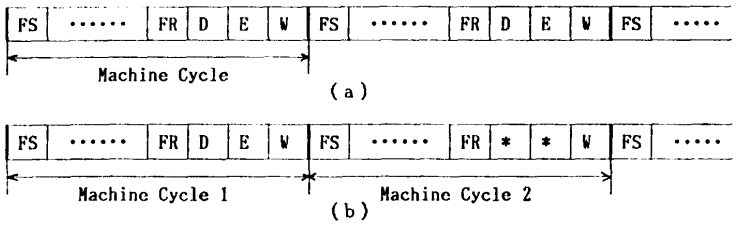

図3 1個の命令流内の命令系列の実行順序
FS: フェッチ発信，FR: フェッチ受信，D: デコード，E: 実行，W: 書き込み，*: アイドリング
(a) レジスタ間演算命令の場合，(b) LOAD 命令の場合

Fig. 3 Execution sequence of instructions in an instruction stream. (a) register-to-register operation, (b) load execution using 2 machine cycles.

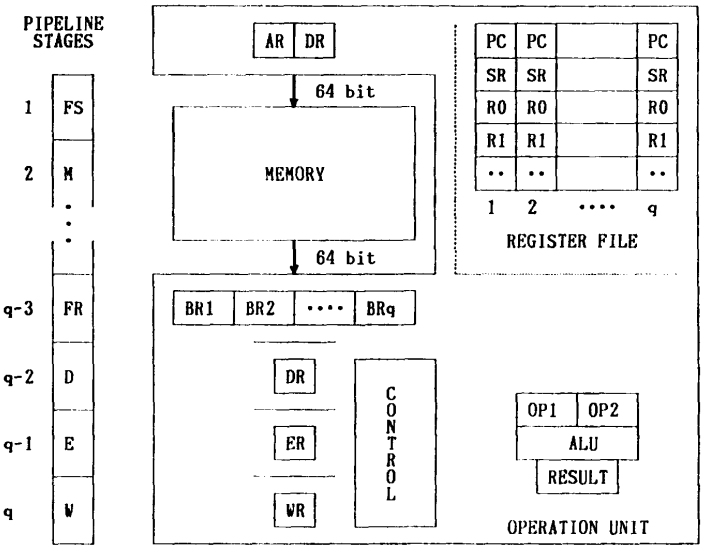

図4 プロセッサの基本内部構成とパイプライン・ステージ
DR: デコード制御レジスタ，ER: 実行制御レジスタ，WR: 書き込み制御レジスタ

Fig. 4 Basic organization of a pipelined MIMD processor and its pipeline stages.
DR: decode control register, ER: execution control register, WR: write control register.

プロセッサ単体の処理能力を上げるためには，並列に動作する多数の演算回路を組み込まなければならない．パイプライン化浮動小数点ユニットには多数の素子を必要とするが，これを格納できると大きな計算能力となる．たとえば，プロセッサのクロック周波数を 25 MHz としても，2クロックで1浮動小数点演算が実行できるので，最大 12.5 Mflops の能力となる．プロセッサ 256 個のマシンで 3.2 Gflops が実現できる．

4.2 キャッシュメモリの併用

この動作原理を採用する場合でも，スループット・

ネックを解消するためのキャッシュメモリは必要である．メモリアクセスにおいて，キャッシュメモリがヒットした場合にはアクセスがただちに完了するが，パイプラインの動作タイミングは変更しない．キャッシュヒットはネットワークに対する要求発信数の節減として利用する．3.2 節の解析から知れるように，ミス率が表3の値以下であれば，平均としては要求が滞留することはない．並列マシンでキャッシュを使用するとキャッシュ・コヒーレンシイの問題が発生するが，この場合にはミス率として 25% 程度が許容されるので，コードとローカル変数のみをキャッシュに格納する方式が採用可能となり，コヒーレンシイの問題を回避できる．

一例として，$s_N=s_M=2$ の場合を考えよう．この場合，主メモリの最大スループットは 0.5 であり，2プロセッサ・クロックごとに1個の要求しか受け付けることができない．命令アクセス要求はまずキャッシュメモリに送られ，ヒットすると読み出し内容がその命令流の入力バッファ・レジスタ BR に書き込まれる．ミスするとその要求は出力バッファに格納され，2クロックごとに1要求がネットワークに出力される．ネットワークから返信が到着するとバッファ・レジスタ BR に書き込まれる．

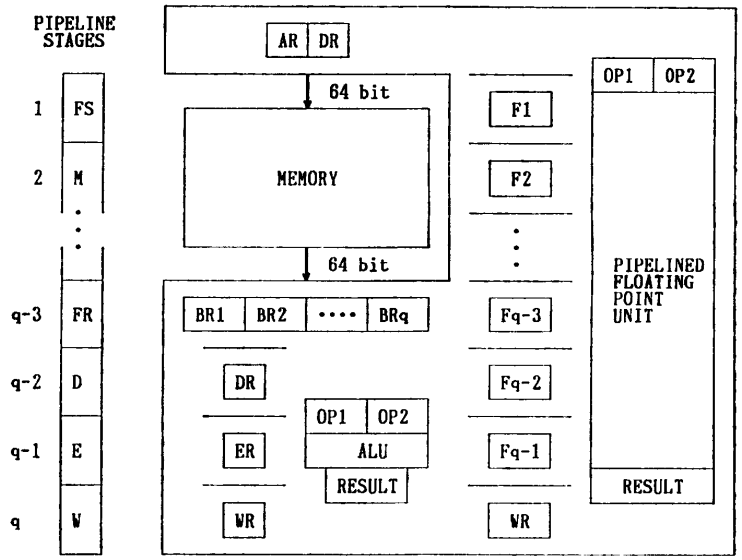

図 5 パイプライン化浮動小数点ユニットの組み込み
第2マシンサイクルの1段～$q-1$ 段を消費して実行
F1, F2, …, F$q-1$: 各段演算制御レジスタ

Fig. 5 On-chip implementation of a pipelined floating point unit.
It works during the 2nd machine cycle.
F1, F2, …, F$q-1$: execution control registers.

```
FS : S1-1,S2-1,S3-1,S4-1,S5-1,S6-1,S7-1,S8-1,S9-1,SA-1,SB-1,SC-1,SD-1,S1-2,
  :      S1-1,S1-1,S3-1,S3-1,S5-1,S5-1,S7-1,S7-1,S9-1,S9-1,SB-1,SB-1,SD-1,
M  :            S1-1,S1-1,S3-1,S3-1,S5-1,S5-1,S7-1,S7-1,S9-1,S9-1,SB-1,
  :                  S1-1,S1-1,S3-1,S3-1,S5-1,S5-1,S7-1,S7-1,S9-1,
FR :                                S1-1,S2-1,S3-1,S4-1,S5-1,
D  :                                     S1-1,S2-1,S3-1,S4-1,
E  :                                          S1-1,S2-1,S3-1,
W  :                                               S1-1,S2-1,
```

(a)

```
FS : S1-1,S2-1,S3-1,S4-1,S5-1,S6-1,S7-1,S8-1,S9-1,SA-1,SB-1,SC-1,SD-1,S1-2,
  :      S1-1,S1-1,S2-1,S2-1,S3-1,S3-1,S7-1,S7-1,S9-1,S9-1,SB-1,SB-1,SD-1,
M  :            S1-1,S1-1,S2-1,S2-1,S3-1,S3-1,S7-1,S7-1,S9-1,S9-1,SB-1,
  :                  S1-1,S1-1,S2-1,S2-1,S3-1,S3-1,S7-1,S7-1,S9-1,
FR :                                S1-1,S2-1,S3-1,S4-1,S5-1,
D  :                                     S1-1,S2-1,S3-1,S4-1,
E  :                                          S1-1,S2-1,S3-1,
W  :                                               S1-1,S2-1,
```

(b)

図 6 $s_N=s_M=2$ の場合のパイプライン実行（ただし，S1 から SD まで 13 個の命令流を実行するとしている）
(a) 1個おきにミスとなった場合，(b) S1, S2, S3 がミス，S4, S5, S6 がヒットの場合

Fig. 6 Pipeline execution for the case of $s_N=s_M=2$, where 13 instruction streams S1-SD are being executed.
(a) miss: S1, S3, S5, S7, S9, …; hit: S2, S4, S6, S8, SA, …,
(b) miss: S1, S2, S3, S7, S9, …; hit: S4, S5, S6, S8, SA, ….

図 6 にパイプライン実行のタイミングを示す．ここでは，S1 から SD までの 13 個の命令流を実行するものとしている．伝送遅延に2クロックの余裕がある．同図(a)は，ミスが1個ごとに発生した場合である．返信未到着によるアイドリングは発生しない．同図(b)は，3個のミスが継続し，次の3個はヒットとなった場合である．2クロックの余裕が置かれているので，この場合もアイドリングは発生しない．もちろん，ミスがより多数継続して発生するとアイドリングが発生し，処理効率が低下する．低速度の主メモリを使用する場合には若干の

効率低下は許容しなければならない.

この方式の場合，多数の命令流が並列に実行されるので，同一ミス率でもキャッシュメモリはより大容量のものが必要になる可能性はある．ただし，もともと25% 程度のミス率が許容されるので，あまり深刻な問題とは思われない．必要となるキャッシュメモリの容量の定量的な検討は今後の課題である．

4.3 命令流数と汎用レジスタの総数

並列実行しなければならない命令流数はかなり大きくなる．N=256, 1024, 4 K, 64 K の場合の計算結果を**表 4** に示す．ここでは，ネットワークの遅延時間としては，最小遅延時間

$$D_{\min}=2ns_N+s_M$$

に 50% の余裕を見て，

$$q=1.5D_{\min} \qquad (9)$$

としている．キャッシュのミス率が所定の値以下となり，ネットワークの負荷率を 75% 以下に保つことができる場合には十分な遅延余裕である．汎用レジスタの総数は，1組のレジスタ数を 32 とする場合の値を示してある．

表 4 から知れるように，必要なレジスタの総数は数千個であり，集積数の点では大きな困難はないと思われる．ただし，数十組のレジスタファイルから1組を選択して1クロックで読み出し書き込みを実行しなければならないので，回路設計には何らかの工夫が必要になる．

4.4 透 明 性

この方式の最大の利点は，多段ネットワークに起因する伝送遅延がプログラマ（およびコンパイラ）から完全に隠蔽されることである．

表 4 ネットワーク段数と命令流数および汎用レジスタの総数の関係（ただし，遅延余裕 50%，1組の汎用レジスタ数 32 の場合）

Table 4 Number of instruction streams and the total size of general purpose register files required a n-stage network machine.

(a) $s_N=s_M=2$

N	n	q	Register file size
256	8	51	1632
1,024	10	63	2016
4K	12	75	2400
64K	16	99	3168

(b) $s_N=s_M=3$

N	n	q	Register file size
256	8	77	2464
1,024	10	95	3040
4K	12	113	3616
64K	16	149	4768

単一命令流のプロセッサ要素を使用する場合には，返信の到着を待たずに要求をつぎつぎと発信する方式を採用する．したがって，一つのプロセッサ要素が複数のメモリ要素にアクセス要求を発信する場合，アクセス要求が発信した順序どおりに実行されることが保証されなくなる．これは，ロックフラグなどの共有変数を用いる同期の実現を複雑化する．通常は，メモリ要素に "Test and Set" あるいは "Fetch and Add" をアトミックに実行する機能を用意して相互排除を実現するが，問題となるのは，この要求を発信してロックフラグをセットし，相互排除領域にアクセスしたあとのロックの解除である．たとえば，一連の STORE 命令の直後にロックフラグをリセットする命令を書いても，すべての STORE 命令がロックフラグのリセットより先に実行されることが保証されない．

しかし，本論文の方式を採用すると，一つの命令流中に出現するメモリアクセスは，かならずプログラムに記述された順序どおりに実行される．一つの命令流においては，ある要求が発信されたあと返信が到着するまでは次の要求が発信されないからである．したがって，共有変数を用いる同期の実現が簡単になる．プログラマは，単一の共有メモリに直接結合された Nq 個のプロセッサをハードウェア資源と考えてプログラミングすればよい．

4.5 命令流の性格選定

ハードウェアとしては命令流の性格に何の制約も課さないが，あるプロセッサ要素では q 個の命令流がハードウェアによるスイッチで並列実行されるので，メモリ管理の負担を考えると，これらの命令流としてはアドレス空間を共有するものが望ましい．有力な候補としてはスレッドが考えられる．

ハードウェアとしては特定の個数の命令流をサポートするが，ソフトウェアとしては任意個数のスレッドを生成して並列実行させてよい．もちろん，各スレッドは相互に同期を取りつつ処理を進行させる．あるスレッドで同期待ちが発生すればこれをサスペンドさせ，実行待ちスレッドから1個を選択して走行させる．

以上より，パイプライン化 MIMD を採用する方式の特徴は，

利点：伝送遅延ネックは完全に解消できる，
　　プログラムからは完全に透明になる，

欠点：プロセッサ要素に高集積度が必要になる，
レジスタセット数十組から1組を選択して1クロックでアクセスする必要がある，

となる．欠点となるのは集積技術の問題であり，その解決に本質的な困難はないと思われる．

5. おわりに

多段結合ネットワークを用いる共有メモリ型超並列マシンの実現においては，経済性を考慮に入れると，プロセッサ要素の動作速度と，多段ネットワークおよびメモリ要素の動作速度の間にある程度のギャップを許容する必要がある．その場合，多段ネットワークとメモリ要素で構成される主メモリのアクセスに関しては，伝送遅延ネックとスループット・ネックを区別して対策を考えるべきである．本論文では，この二つのネックに起因する効率低下を防止する手段を検討し，キャッシュメモリを単独で使用する方式では伝送遅延ネックを解消するのは困難であることを示し，パイプライン化 MIMD プロセッサとキャッシュメモリを併用する方式を提案した．

現在，プロトタイプ機として 16 命令流，メモリ管理ユニット内蔵のプロセッサの論理設計を進めている．

謝辞 討論いただいた東京大学工学部計数工学科出口光一郎，田胡和也，永松礼夫の諸氏に謝意を表する．

参考文献

1) Gottlieb, J. et al.: The NYU Ultracomputer—Designing of an MIMD Shared Memory Parallel Computer, *IEEE Trans. Comput.*, Vol. C-32, No. 2, pp. 175-189 (1983).
2) Pfister, G. F. et al.: The IBM Research Parallel Processor Prototype (PR 3), *Proc. Int'l Conf. on Parallel Processing*, pp. 764-771 (1985).
3) Growth, W. et al.: Performance Measurements on a 128-node Butterfly Parallel Processor, *Proc. Int'l Conf. on Parallel Processing*, pp. 531-535 (1985).
4) Flynn, M. J.: Some Computer Organizations and Their Effectiveness, *IEEE Trans. Comput.*, Vol. C-21, No. 9, pp. 948-960 (1972).
5) Smith, B. J.: A Pipelined, Shared Resource MIMD Computer, *Proc. Int'l Conf. on Parallel Processing*, pp. 6-8 (1978).
6) Jordan, H. F.: Performance Measurements on HEP—A Pipelined MIMD Computer, *Proc. 10-th Annual Int'l Conf. on Computer Architecture*, pp. 207-212 (1983).
7) 後藤英一ほか：FLATS 2 のアーキテクチャ，第 35 回情報処理学会全国大会論文集，6C-4, pp. 191-192 (1987).
8) 市川周一ほか：循環パイプライン・アーキテクチャ—pipelined MIMD の試み—，第 37 回情報処理学会全国大会論文集，4N-4, pp. 123-124 (1988).
9) 田胡和也ほか：多段結合ネットワークを用いた共有メモリ型マルチマイクロプロセッサの設計，第 38 回情報処理学会全国大会論文集，7T-1, pp. 1494-1495 (1989).

(平成元年5月30日受付)
(平成2年2月13日採録)

森下 巖（正会員）

1934年生．1957年東京大学工学部応用物理学科計測工学コース卒業．同年東レ(株)入社，計装制御システムの設計に従事．1966年東京大学工学部計数工学科助教授，1980年同教授，現在に至る．工学博士．1973年 SRI 人工知能研究センタ滞在．パターン認識，信号処理，画像処理，マルチプロセッサシステムなどの研究に従事．著書「マイクロコンピュータのハードウェア」(岩波書店)，「マイクロコンピュータの基礎」(昭晃堂)，「信号処理」(計測自動制御学会) など．IEEE，計測自動制御学会，電子情報通信学会，電気学会などの会員．