マルチマイクロプロセッサの応用

APPLICATION OF MULTIMICROPROCESSOR

並列パイプラインモジュール　ＰＰＭ

PARALLEL AND PIPELINE MODULE (PPM)

阿江　忠　　相原玲二　　飯田　優　　佐藤輝久

Tadashi Ae　Reiji Aibara　Masaru Iida　Teruhisa Sato

広島大学　　工学部

Faculty of Engineering , Hiroshima University

1　まえがき

マルチマイクロプロセッサは、LSI/VLSI技術の進歩により多数個プロセッサシステムまたはモジュールとしての実用化を迎えようとしている。　一方、マルチプロセッサという形態は古くから構想や製作がなされている(1)。しかしながら、ネットワークという分散システムとしての進展を除けば、並列処理システムの普及はアレイプロセッサなど数えるほどにすぎない。　我々も、マイクロプロセッサが安価に出回るようになってから実験システムの試作などを行い、特に、並列処理システムとしての問題点を種々の角度から検討した(2)~(9)。その結果、結合形態はモデル論からはいろいろ考え得るとしても、プロセッサ間通信が実用上のネックの主因であることから、目的に合った構造を如何に見いだすかが重要である。

現在、この意味では、ＦＦＴの演算、偏微分方程式を解くためのアレイプロセッサなどが、市販あるいは試作機として実用化の段階にある。だが、明らかにこれらアレイプロセッサも専用モジュールであり、目的ごとに種々の形態を検討することが必要であろう。

本稿では、一つの有用な結合形態として並列パイプライン構造を提案し、その特長を述べる。並列パイプライン構造は、スター状の並列プロセッサと線状パイプラインプロセッサの融合形であり、製作の容易な構造である。　数年前に製作ずみの実験機ＡＫＯＶＳＴはこのシミュレーションを行なえる形態になっており(6)、まず、並列パイプライン構造とその単位プロセッサであるプロセッサユニットについて述べたのち、応用について一般的な議論をする。　しかるのち、現在製作中の並列パイプラインモジュールＰＰＭを念頭においた応用例　ソート／サーチについて述べ、ＰＰＭの具体的な回路についても触れる。

2　並列パイプライン構造と単位プロセッサ

本稿で対象とするマルチプロセッサは図１のような構造（以下、並列パイプライン構造をもつモジュールをＰＰＭと略す）をもつ。　マルチプロセッサは多数個のプロセッサの一次元線状結合であり、単位プロセッサを以下ではプロセッサユニット（ＰＵ）と呼ぶ。　各プロセッサユニットは、ホストコンピュータと、インタ

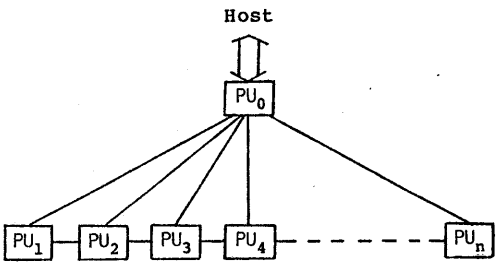


図１　並列パイプライン構造をもつモジュール　ＰＰＭ

ープロセッサ PU_0 を介して結合される。PU_0 からプロセッサユニット $PU_1 \sim PU_n$ へは、すべて結合があり、かつ、PU_i と PU_{i+1}（$i = 1, 2, \cdots, n-1$）の間に結合がある。いずれの結合も双方向性である。プロセッサユニット PU_i の入出力は図2のように表される。

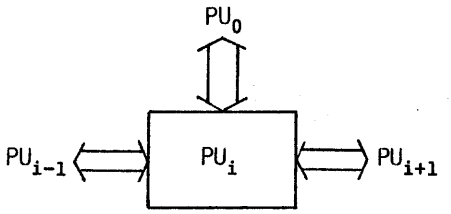


図2　単位プロセッサの入出力

このような双方向3ポートをもつPUはかなり融通性をもつ†。一般に、このような多ポートをもつPUの本来の働きをリアルタイム処理させるには、マルチタスク化が望まれるが[10]、ここではPUの機能はさほど大きくせず、むしろ、PU数の上限を制限しない方向をとる。従って、PUの動きは、マクロには図3のような状態遷移で表される。

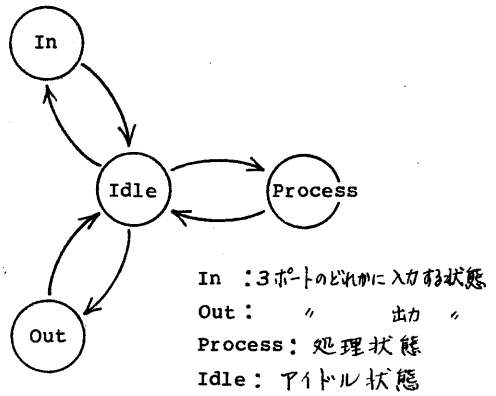


図3　単位プロセッサの状態遷移（マクロ）

†）著者らは、インハウスネットワークのノードプロセッサとしてもこのようなプロセッサユニットを使っている[10]。

PUの機能は、具体的にはプログラムにより指定される。各PUには、ほぼ均一な機能をもたせるが、若干のプログラムの違いにも対応できる††。また、いくつかのプログラムを切換えることで、複数機能をもつPUとして使うこともある。

3　並列パイプライン処理

本節では処理の形態について議論する。ここでは、本質を明らかにするため、プロセッサユニットをノード、結合をラインで表した有向グラフを対象とする。ちなみに、我々の提案するPPMは図4のようなグラフになる。（a）は物理的な構造のまま表現しているが、（b）はインタープロセッサ PU_0 の入出力を分離したノードで表し、論理的な結合は、以下（b）のようなグラフで考える。

まず、並列パイプライン処理の基本となる並列処理とパイプライン処理について述べる。

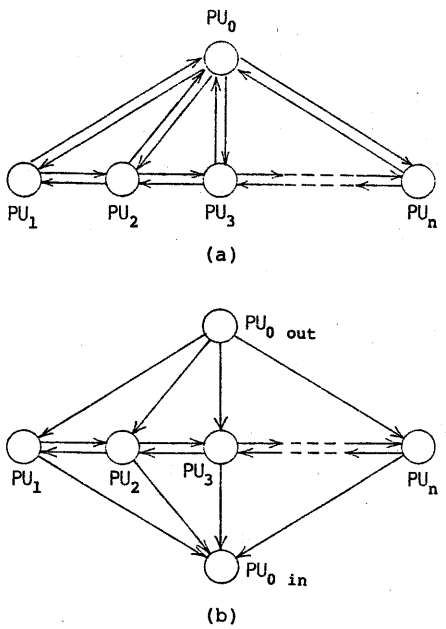


図4　PPMのグラフ表現

††）本来、各PUが全然異なった機能をもってもよいが、設計が面倒すぎる。

並列処理

並列処理は一般に図5のように表現できる。$PU_{0\,out}$ における データの集合が分割され、$PU_1 \sim PU_n$ にそれぞれが転送されれば、それらの処理は並列に実行される。 この処理形態はごく一般的なものであり、多くのマルチプロセッサがこの処理形態をもつ。 我々の実験システム AKOVST の並列処理も、当初、この処理の実験のために製作され、画像表示を対象とした種々のデータが得られている(3)~(6)。

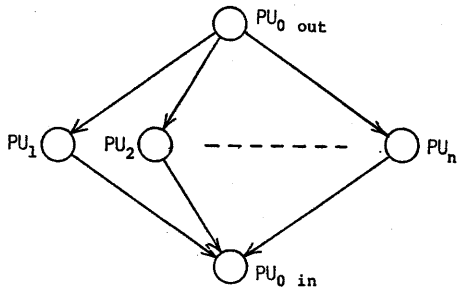


図5　並列処理

尚、並列処理における $PU_{0\,out}$ から各PUへのデータ転送は、逐次的なもの以外に、同一データの一せい転送（ブロードキャスト）のモードをもつことが望まれる†。

パイプライン処理

パイプライン処理は、一般に、図6のように $PU_{0\,out}$ から出たデータの流れが PU_1, PU_2, … と順次経由して $PU_{0\,in}$ に至るような処理形態をとり、同時刻に（プロセッサユニットは異なるが）複数個のデータの処理がなされる。

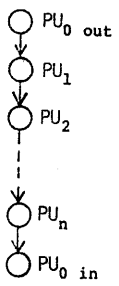


図6　パイプライン処理

†) 試作中の PPM はブロードキャストの機能をもつ。

並列パイプライン処理

本稿では、以上の並列処理とパイプライン処理をあわせもった図7のようなものを並列パイプライン処理というものとする。

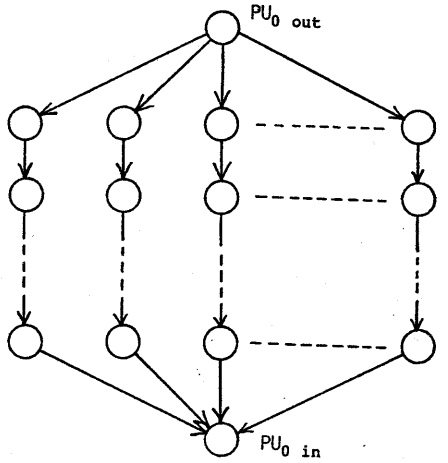


図7　並列パイプライン処理

縦方向のデータの流れは相互に関連し合わない点が、アレイ状の処理と異なる。

図7の並列パイプライン処理は並列処理の一つの形態であり、構造のみに着目した処理速度については文献(9)で検討した††。

ところで、我々の提案する構造（PPM）が、並列処理、パイプライン処理はむろん、図7の並列パイプライン処理を実現できることは、グラフの埋込みからすぐ判明する。（図8参照）すなわち、図4（b）のPPMの処理形態は、見た目よりかなりの普遍性をもつものと考えられる。

††) 処理速度の点でベストではないが、考え易い形として一般性がある。 処理速度の優れた形態としては、入れ子並列型、ブロードキャスト型などがある(9)。

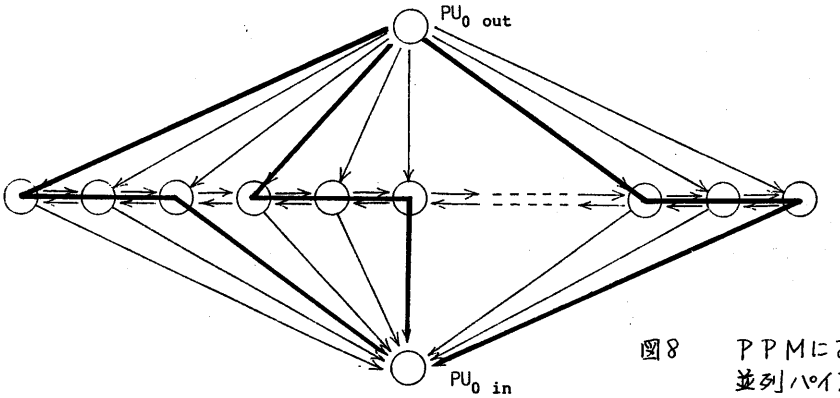


図8　ＰＰＭにおける
並列パイプライン処理

4　応用概観

応用例は、ＰＰＭの処理形態にあわせて種々考えられる。

並列処理

図５のような処理で、行列演算をはじめとする種々の分割可能な演算が相当する。　その応用として、画像処理における種々の演算（３Ｄ→２Ｄ変換、スプライン曲線の計算など）が挙げられる。　一例として、プロセッサユニット数に対して処理速度が向上する様子を実験機ＡＫＯＶＳＴにおいて測定したものを図９に示す(6)。

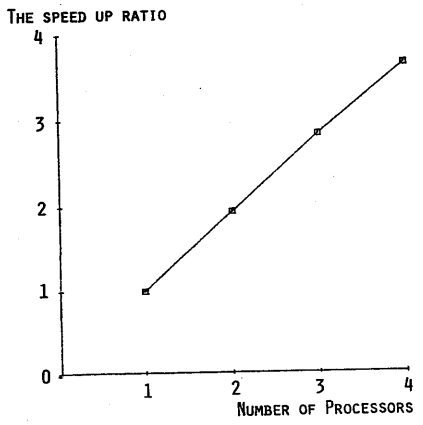


図9　スプライン関数によるスムージング"における処理速度向上の実測値

パイプライン処理

図６のようなパイプライン処理は、典型的な応用として、複数個のパターンとのマッチングを調べる場合などに有効である。

一方、パイプライン処理は、図６のような一方向の場合のみではなく、図10のような二方向パイプラインも有用である。（二方向の場合は便宜上横の流れとして描く。）

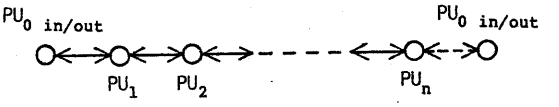


図10　二方向パイプライン処理

二方向パイプライン処理の場合、左端のPU₀が入出力に携わるならば、右端のPU₀は必ずしも必要でない。

二方向パイプライン処理は、その名の如く処理がスムーズに行なえるためには、各ＰＵの動作が特定の条件を満たしている必要がある†。一般的に、プロセッサユニットは、互いにハードウェア的には同期機構をもたない。　従って、一方向では楽な動作も二方向では必ずしも容易でないことが多い。

ここで、二方向パイプラインの簡単な一例としてリニアスタック（以下ＬＳと略す。）を取り上げる。　ＬＳの動作の概略は、図11のように、

† Kung のいう "systolic computation" に相当。

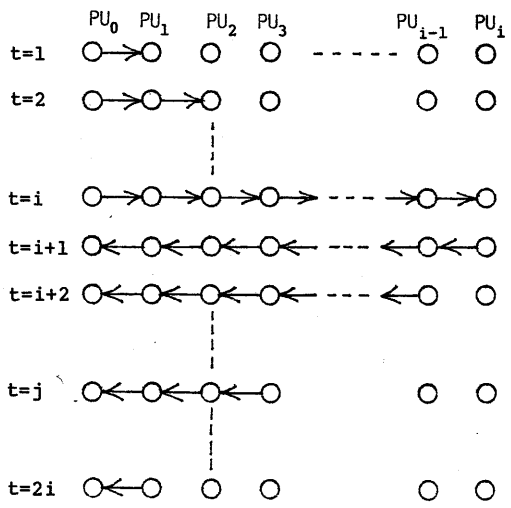


図11　LSの二方向パイプライン動作

時刻 $t=0$ からある時刻 $t=i$ までは右側への一方向パイプライン動作を行ない、$t=i+1$ からある時刻 $t=j$ までは、逆転して、左側への一方向パイプライン動作をする†。　このあと、再び右方向パイプラインへと移ることもあり得る。つまり、右方向パイプラインがスタックへデータを詰め込む操作であり、左方向パイプラインがスタックからデータを取り出す操作に相当する。

LSでは、図11のように右側への一方向パイプラインから左側への一方向パイプラインの切換えが、スムーズにムダなくできる必要がある。

ここで、データの詰め込みを［phase 1］、取り出しを［phase 2］と称している。

一般にスタックはポインタを用いたソフトウェア的なものが普通であり、このようなハードウェアスタックはあまり使われない。　しかし、各プロセッサユニットごとにある程度の処理が可能であり、有用な場合もある。　その一例は文献(11)で述べたソーティングモジュール‡であり、詳しくは次節で述べる。

†) むろん、$0<i<j$。全PU数を n とすると $i<n$。

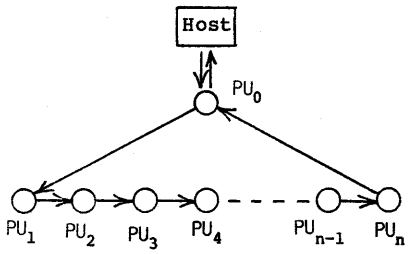
‡) 文献(11)ではデータの入出力は考えていないので、その部分を付け加えて考える。

以上は、並列処理、パイプライン処理別々に応用を考えてきたが、前節で述べたようにこれらを混合した並列パイプライン処理（図ワ）はさらにそれぞれの長所をあわせた処理形態となり得る。　また，並列処理とパイプライン処理が時間的に切換わるような処理形態もとられる。

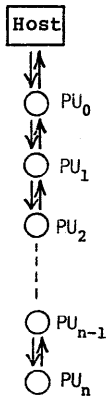
5　ソート／サーチへの応用

PPMを用いる応用例の一つとして、ソート／サーチについて述べる。　まずソートについて考える。　ここでのソートは、ホストコンピュータからインターフロセッサPU₀を介してデータ列が入り、昇順または降順にソートされたデータ列が出てくるような操作、すなわち、入出力手続きを考慮したソートを対象とする。

このとき、ホストコンピュータからみたソーティングモジュールの使い方を、データ列の入出力から大別すると、図12（a）のFIFO型



(a)　FIFO型



(b)　LIFO型

図12　ソーティングモジュールの使い方

と (b) のＬＩＦＯ型になる。　ここでのＬＩＦＯ型は前節のＬＳで十分である。　また、ＰＵの個数に制限をつけなければ、ＬＳはＦＩＦＯ型に変換できる†。　ＰＵ数が十分用意されたＰＰＭを用いれば、ＦＩＦＯ型、ＬＩＦＯ型いずれのソーティングモジュールを作ることもできる。

図13のＰＰＭによるＬＳを用いるソートを簡単に述べる。（昇順ソートを例にとる）

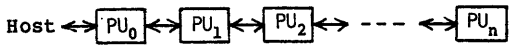


図13　ＰＰＭによるＬＳ

[phase 1] では、ＰＵは左から入ってくる入力 input と手持のデータ register とを比較し、大きい方の値を右隣りのＰＵへ output とし出力する。　[phase 1] が終った時点で左から昇順に並んでいる。　[phase 2] ではデータを左へ転送し出力する。　Algol風には次のように書ける。

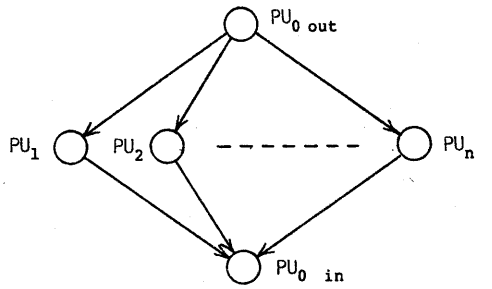
```
[ phase 1 ]
    input ;    / leftport /
    output ;   / rightport /
    register ; / data storage /
begin
    register:=NIL ;   / initialize /
    while phase1 do
        if input>=register
            then output:=input
            else begin
                    output:=register ;
                    register:=input
                 end
end

[ phase 2 ]
    input ;    / rightport /
    output ;   / leftport /
begin
    while phase2 do
        begin
            output:=register ;
            register:=input ;
        end
end
```

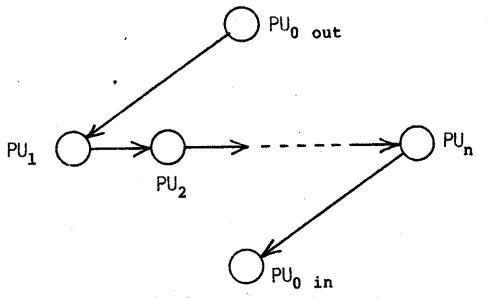
このＬＩＦＯ型ソートをＦＩＦＯ型に変更しようとすると、昇順と降順が逆になるだけである。　一般にＦＩＦＯ型はデータ列の長さがＰＵ数を決めるから、固定した構造ではＰＵ数が大きすぎると遅延時間が問題になる。　しかし、ＰＰＭでは構造上このことも問題にならない。　データ列の長さに相当するプロセッサユニット PU_i は PU_0 に直接結合され得るからである。

実際にこのアルゴリズムをインプリメントし、スムーズに動作させるには、[phase 1] と [phase 2] の切換えやデータ通信のための配慮が必要である。　また、降順のソートも同様にできる。

次にＰＰＭによるサーチについて考える。データ数以上のＰＵが用意できる場合、そのサーチは非常に簡単である。　ＰＰＭでは、図14 (a) のようにブロードキャストを用いてのサーチ、(b) のようにパイプラインによるサーチを行なえる。　これらいずれの場合も、キーあたり Ｏ(1) の手数ですむ。



(a)　並列サーチ



(b)　パイプラインサーチ

図14　ＰＰＭによるサーチの方法

以上、ＰＰＭによるソート／サーチを、ＰＵ数がデータ数を下回らないという条件で考えてきた††。　このことは、ＰＵの複雑さの程度から

† 一般に、ＬＩＦＯ型は、ある条件のもとではＦＩＦＯ型に変換できる。

†† このようなソートのうち入出力手続きを考えないものは並列バブルソート[12]として知られており、入出力手続きも考え、バスによるブロードキャストを利用したソート[13]も最近提案されている。

ら考えて、たとえVLSI技術を用いてもきつい条件といえよう。また、データ数nに等しいPUが用意できる場合、ソートの速度はFIFO型もLIFO型も似たようなものである。もっとも、ソートすべきデータ列の集合が多くあり†、それらを順次ソートする場合などには、FIFO型ではそのままでマクロにもパイプライン動作ができるが、LIFO型では工夫が必要となる。

データ数nだけPUを用意するということは、1つのPUにデータ格納レジスタは1個だけである。ここで考えるPUは、一通りの演算機能（この例では、比較）をもち、かつプログラマブルとしている。従って、1つのPUあたりデータ格納レジスタ1個は‡PUの複雑さに比べて相対的に非常に小さい割合でしかない。1PU1データという構造はスペース利用率の悪いものとなっている。

では、1PUkデータとしたとき、

(i) $k \geq 2$でも$k=1$と同じ速さのソート／サーチは実現可能か？

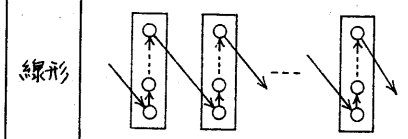
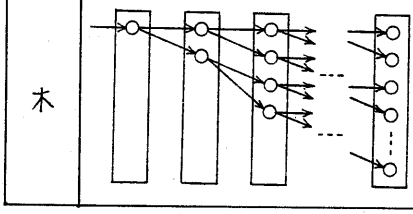
(ii) 可能とすれば、kの値はいくらが適当か？

という問が生ずる。

「一般のkに対して、どのような速さになるか」は、PUに格納する全体のデータ構造に依存する‡‡。ソートに有効な、PUとデータ構造の関係を図15に示す。

線形データ構造の場合、kが2以上における処理速度と、kが1の処理速度を大差なくするのは非常に困難である。

一方、木型データ構造ではFIFO型ソートは困難であるが、LIFO型ではデータを入力する段階で取り出しに向いた格納方法をとり、データを出力するときにソートされたデータ列にするという方法が知られている。この方法を用い、セル数$O(\log n)$で実現した文献(15)は、PU数の下限が小さくなるという実用性の非常に高い方法である‡‡‡。この場合の木の作り方はいろいろあり、全ノード数がデータ数を上回る2分木という条件さえ満たしていればよい。1つのPUに格納するデータ数を極力同じにし、各PUでの処理をほとんど同じとし、かつ、データのアドレスを処理に便利なものとするため、我々は図16のようなデータ構造を採用した‡‡‡。

タイプ	PU全体でのデータ構造	備考
線形		FIFO LIFO 両方可
木		LIFO のみ

注：□は1つのPUをさす。

図15　PUとデータ構造の関係

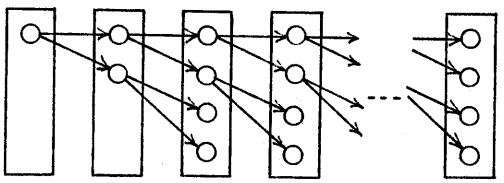


図16　あるPUからkが一定となる木構造

ただし、図16は$k=4$の例を示している。

このようなデータ構造のとき、PU数nとデータ数n_dの関係は次のようになる。（ここで、$k=2^l$：lは正の整数とする）

$$n_d=(2k-1)+k\{n-(\log_2 k+1)\}$$
$$=k(n+1-\log_2 k)-1$$

†) テーブル型データの桁ごとのソートなど。

‡) これに、さらにワーキングレジスタが1個加わる。

‡‡) $k=1$の場合は他に、ブロードキャスト型(13)やデータレジスタと比較器と分離した形式(14)などがある。

‡‡‡) このようなソート専用モジュールを文献(15)ではソートエンジンと呼んでいる。

‡‡‡) 文献(15)では、木を縦続に接続する方法を提案している。

逆に、

$$n=\begin{cases}\left\lceil \frac{n_d+1}{k} \right\rceil - 1 + \log_2 k & (n_d \geq 2k) \\ \left\lceil \log_2 (n_d+1) \right\rceil & (n_d < 2k)\end{cases}$$

例えば、データ数が10000、kが1024のとき必要なPU数nは上式より $n=19$ と求まる。

このようなLIFO木†によるソートに要する時間は、$k=1$, すなわちデータ数だけPUを用意するソートの場合と同程度となる。 1PUあたり2以上のデータを格納するにもかかわらず、1PUあたり1データの場合と処理速度が変わらないのは、木構造の性質をうまく使っているためである。 このアルゴリズムを以下に示す。 (昇順のソートを示す)

```
[ phase 1 ]
    input ;    / leftport /
    output ;   / rightport /
    register[k] ;   / data storage /
begin
    for count:=0 while phase1 do
        begin
            address:=calculate(count) ;
            if input>=register[address]
                then output:=input
                else begin
                        output:=register[address] ;
                        register[address]:=input
                     end
        end
end

[ phase 2 ]
    input ;    / rightport /
    output ;   / leftport /
    addinput ;   / leftport /
    addoutput ;  / rightport /
begin
    while phase2 do
        if register[2*addinput]<=register[2*addinput+1]
            then begin
                    output:=register[2*addinput] ;
                    register[2*addinput]:=input ;
                    if 2*addinput<k
                        then addoutput:=2*addinput
                 end
            else begin
                    output:=register[2*addinput+1] ;
                    register[2*addinput+1]:=input ;
                    if 2*addinput+1<k
                        then addoutput:=2*addinput+1
                 end
end
```

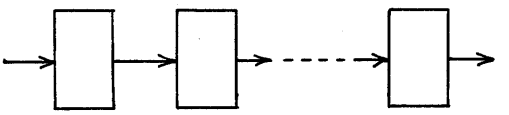
[phase 1]では、入ってくるデータを、木の各ノードへ左のPUから順に詰めてゆく操作がなされる。 ただし、「親よりもまるのデータの方が小さくない」という関係を常に保つようにデータの比較を行ないながら木を構成してゆく。 なお、左のPUから入力されたデータが、そのPU内のどのデータと比較すべきかということは、PU内に入力(通過を含む)されたデータ数から計算して決める。 このアルゴリズムやcalculate(count)がそれにあたる。

[phase 2]では、まず、左隣のPUからのアドレス情報(さらに左のPUへ出力したデータの格納アドレス)を入力し、その子にあたるデータ(2つ)のアドレスを計算する。 それら2つのデータを比較し、小さい方を左のPUへ出力する。 また小さいデータの格納アドレスを右のPUへ出力する。 各PUがこの操作をすることにより、昇順にソートされたデータ列を出力することができる。 降順のソートも同様に行なうことができる。

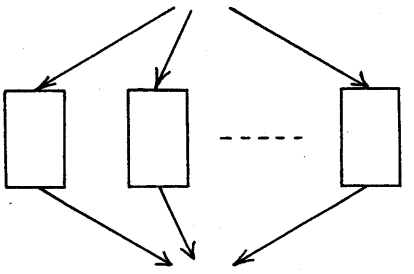
このような、LIFO木によるソートの特長は、本質的にはPU数が速度に関係しないから、スペースだけの観点で1PUに格納するデータ数kを決定できる††。

次にサーチについて考える。

kが2以上の場合でも、やはり、図17(a)のようにパイプライン的なサーチと(b)のよう



(a) パイプラインサーチ



(b) 並列サーチ

図17 PPMによるサーチ ($k \geq 2$)

† 文献(15)ではヒープ木。

†† 最適なデータ数kは、演算部のプログラムを含むハードウェア量から算定できよう。恐らく数百のオーダではないかと推定している。

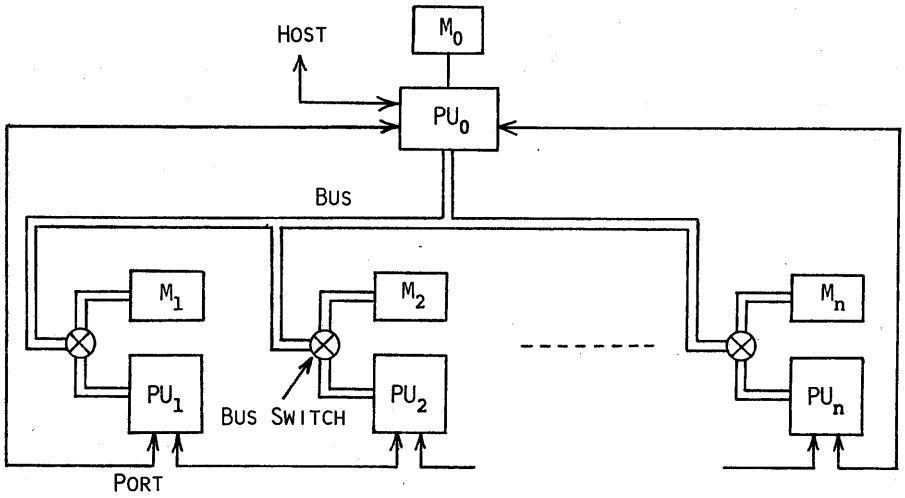


図18　　PPM の構成

な並列サーチが考えられる。いずれの方法もPPMでは可能である。

ソートを行ない、その後にデータを入れ換えずそのままサーチに移ると仮定すると、線型データの場合、kがある程度大きくなっても並列サーチは効果があがるものと思われる。LIFO木の場合は、データの並びが一列でないため線形データ構造ほどの効果は期待できない。しかし、PU数が多いときにブロードキャストを用いれば、並列サーチは通常の意味での並列処理の効果はあろう。

また、ソートとサーチが別個になされてよいならば、文献(15)のようにソートのあとで入れ換えをしてサーチに移れば、データを木構造としたパイプラインサーチも有効となる†。

いずれの使い方をするかは、アーテクチャ的にはPPMのPU数に依存し、ソフトウェア的にはどのような処理を対象とするかに依存しよう。いずれにしても、OS設計で十分に考慮されなければならない。

6　製作中のPPM

現在製作中のPPMの構成を図18に示す。PU_0～PU_nは、いずれもCPUにマイクロプロセッサ Zilog Z-80A を使用している。また、M_0～M_nは最大12KBのメモリである。各PUはバスにより結合されており、M_i ($i=1,2,\cdots,n$) はPU_0とPU_iの共有メモリと見ることができる。M_1～M_nに対するPU_0のアクセスには2つのモードがあり、それらは表1のようになっている。M_iに対してPU_0とPU_iがアクセスするときの優先権は、モードIのとき同等であり、モードIIのときはPU_0の方が高くなっている。

隣接するPU間（PU_0-PU_1, PU_1-PU_2,…等）の結合は、専用LSIを用いたポート結合とな

表1　　M_1～M_nに対するPU_0のアクセスモード

モード	アクセス動作	備考
I	M_1～M_nのうち指定した1つだけをアクセスする	読み出し書き込み共に可能
II	M_1～M_nを一せいにアクセスする（ブロードキャスト）	書き込みのみ可能

†) サーチ用のモジュールを文献(15)では サーチエンジンと呼んでいる。

っている。

製作中のPPMはn=32を実装単位となるよう設計しているが、接続線を用いることでnは本質的には制限なく大きくすることもできる。nが100や1000のオーダでのPPMの効果的な使用法が見出されれば、そのような実装も可能である。

7 むすび

マルチマイクロプロセッサの有効性はすでにいくつかの論文でも確かめられている(16)~(18)。それらの基本的な構造はスターあるいはアレイである。この他、ループや木構造などの試作報告があるが、いずれも適用範囲はそれ程明確でない。

本稿では、並列パイプライン的な動作の可能なスターと線状(またはループ)を組み合わせた構造をもつマルチマイクロプロセッサPPMについて考察した。

PPMの特徴は、構造が簡単なことと拡張性に富むことである。また、過去に製作ずみのシステムAKOVSTも当初はスター状の結合のみであったが、後にループ結合が追加され、PPMとしても動作する。

もっともAKOVSTは集積度の低いチップによる製作であるので、インタープロセッサ以外のPUは4個しかない。従って、現在は32個のPUをもつPPMを製作中であり、間もなく実装される予定である。本稿では、応用としてソート/サーチを中心に述べたが、タポートのチャネルプロセッサ的な使い方など種々の実用的な用途も考慮されている。現状では、小規模なミニコン程度のものに付随するモジュールとして設計されているが、カスタムVLSIとしてさらに集積化される方向も考えられる。

文献

(1) P.H. Enslow Jr. (Ed.): "Multiprocessors and Parallel Processing", John Wiley & Sons, Inc. (1974)

(2) 阿江ほか: "小規模マルチマイクロプロセッサシステムの一方式", 信学技報 EC 78-35 (1978)

(3) 阿江, 高橋, 千葉, 松本: "マルチプロセッサによるグラフィックターミナルの高速化について", 信学技報 IE 80-39 (1980)

(4) T. Ae and K. Takahashi: "Picture data processing in small parallel computer", Proc. 2nd IEEE Workshop on PDDM, pp. 266-271, Asilomar (Aug. 1980)

(5) T. Ae et al.: "Computer graphics in multiple microprocessor system", EUROGRAPHICS 80 (edited by C.E. Vandoni), pp. 281-288, North-Holland Pub. Co. (1980)

(6) 阿江ほか: "ハイレベルデータフロー形並行処理計算機の検討", 信学技報 EC 80-68 (1981)

(7) T. Ae et al.: "A multiple microprocessor system with mutually diagnosing capability", Proc. of International Computer Symposium (Vol. I), pp. 375-388 Taipei (Dec. 1980)

(8) 阿江ほか: 情処学会マイクロコンピュータ研資 18-3 (1981)

(9) 阿江, 松本, 相原: "通信オーバヘッドを考慮した並列処理ネットワークにおけるフロー問題", 信学技報 AL81-31 (1981)

(10) 阿江, 尾崎, 天満: "C言語によるLCNノードプロセッサのソフトウェア開発", 信学会情シ部門全大 586 (1981)

(11) 阿江, 相原: "VLSI向きアルゴリズムの一考察", 信学会情シ部門全大 53-7 (1981)

(12) A. Mukhopadhyay and T. Ichikawa: "An n-step parallel sorting machine", Tech. Rep. 72-03, The University of Iowa (1972)

(13) 安浦, 高木: "並列計数法による高速ソーティング回路の設計", 信学技報 AL80-76 (1981)

(14) D.T. Lee, Hsu Chang and C.K. Wong: "An on-chip compare/steer bubble sorter", IEEE Trans. Computer, C-30, 6, pp. 396-404 (1981)

(15) Y. Tanaka et al.: "Pipeline searching and sorting modules as components of a data flow database computer", Information Processing 80 (edited by S.H. Lavington), pp. 427-432 North-Holland Pub. Co. (1980)

(16) R. Kober et al.: "SMS-A powerful parallel processor with 128 micro computer", Euro Micro Jour., 5, P.98 (1979)

(17) 星野力: "偏微分方程式解析のためのマイクロプロセッサ複合体", 情報処理, 20, 11, p.974 (1979)

(18) 同: "超高速計算技術の動向と原子力分野での試み", 電気学会全大 S.14-1 (1981)