二次元記憶を用いた連想処理システム

ASSOCIATIVE PROCESSING SYSTEM USING TWO DIMENSIONAL ACCESS MEMORY

元岡 達　　田中英彦　　上森 明　　鈴木達郎
Tohru MOTO-OKA　Hidehiko TANAKA　Akira UEMORI　Tatsuo SUZUKI
東京大学工学部
Faculty of Engineering, University of Tokyo

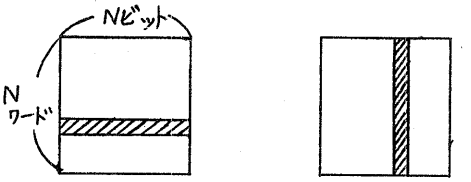
[1] はじめに

従来から、種々の連想プロセッサが提案されてきたが、実際に製作された例は非常に少ない。ここでは二次元アクセス可能な記憶と、多数の処理素子(Processor Element。以下PEと略す。)とから成る連想処理システムの構成と処理内容について述べる。最近進歩の著しいマイクロプロセッサを用いれば、ここで述べるシステムの実用化は容易と考えられる。このシステムの応用として、パターン認識や、データベース管理等の問題向きのプロセッサを目標としている。

[2] 二次元アクセス記憶の構成法

連想処理とは、簡単な比較操作を各記憶セルで並列に行ない、高速に検索する処理である。連想処理に必要な連想記憶は、記憶内容によって、そのアドレス又はデータを取り出すものである。従来から、超伝導素子、磁気薄膜等で連想記憶を構成する方法が提案されているが、大規模で低価格な物は実現されていない。そこで、通常のRAM(Random Access Memory)のアドレスデコード回路を工夫し、2次元アクセス性を持たせた記憶を作り、それに比較操作のロジックを付加する事が考えられた。(2) 図1に示すような、行方向、列方向のアクセス可能な記憶は、データを適当に配置する事によって、容易に実現できる。

図1．2次元アクセス記憶のアクセスモード



(a)ワードスライスアクセス　(b)ビットスライスアクセス

(a)－ある語番号のNビットをまとめてアクセス。
(b)－すべての語のあるビット位置の部分だけをアクセス。

今、N語×1ビットのRAMICを用いて、図1の2次元記憶を構成する事を考える。図1のアクセスが可能となるための条件は、

1．ある語(行)に属する各ビットは、同じメモリーチップ内に存在してはならない。

2．すべての語の同じビット位置(列)に属する各ビットは、同じメモリーチップ内に存在してはならない。

3．アドレスの変換論理が簡単である事。

1回のアクセスで、各チップから1ビットずつ、全体でNビットの入出力が行われるので、上

のようなデータの重複を避ける条件が必要である。上記の条件を満足するような方法として、

① Skewed Array方式

② EOR Skewed Array方式

がある。（図2参照）

図2．N＝8の場合のデータ配置法

① Skewed Array方式

チップ番号	チップ内アドレス 0	1	2	3	4	5	6	7	アドレスの与え方
0	0_0	1_1	2_2	3_3	4_4	5_5	6_6	7_7	$A_1A_2A_3$
1	7_0	0_1	1_2	2_3	3_4	4_5	5_6	6_7	$A_1A_2A_3+1$
2	6_0	7_1	0_2	1_3	2_4	3_5	4_6	5_7	$A_1A_2A_3+2$
3	5_0	6_1	7_2	0_3	1_4	2_5	3_6	4_7	$A_1A_2A_3+3$
4	4_0	5_1	6_2	7_3	0_4	1_5	2_6	3_7	$A_1A_2A_3+4$
5	3_0	4_1	5_2	6_3	7_4	0_5	1_6	2_7	$A_1A_2A_3+5$
6	2_0	3_1	4_2	5_3	6_4	7_5	0_6	1_7	$A_1A_2A_3+6$
7	1_0	2_1	3_2	4_3	5_4	6_5	7_6	0_7	$A_1A_2A_3+7$

② EOR Skewed Array方式

チップ番号	チップ内アドレス 0	1	2	3	4	5	6	7	アドレスの与え方
0	0_0	1_1	2_2	3_3	4_4	5_5	6_6	7_7	$A_1A_2A_3$
1	1_0	0_1	3_2	2_3	5_4	4_5	7_6	6_7	$A_1A_2\overline{A_3}$
2	2_0	3_1	0_2	1_3	6_4	7_5	4_6	5_7	$A_1\overline{A_2}A_3$
3	3_0	2_1	1_2	0_3	7_4	6_5	5_6	4_7	$A_1\overline{A_2}\overline{A_3}$
4	4_0	5_1	6_2	7_3	0_4	1_5	2_6	3_7	$\overline{A_1}A_2A_3$
5	5_0	4_1	7_2	6_3	1_4	0_5	3_6	2_7	$\overline{A_1}A_2\overline{A_3}$
6	6_0	7_1	4_2	5_3	2_4	3_5	0_6	1_7	$\overline{A_1}\overline{A_2}A_3$
7	7_0	6_1	5_2	4_3	3_4	2_5	1_6	0_7	$\overline{A_1}\overline{A_2}\overline{A_3}$

ただし、I_jはWORD番号Iのjビット目を意味する。$A_1A_2A_3$は、WORD番号の2進数である。

図2において、N語×Nビットの行列a_{ij}（i，j＝0，1，---，N－1）の配置法は、以下の通りである。

①の方式（図2の①）

- $i \leqq j$の時、第（$j-i$）番目のチップの第jビット目にa_{ij}を入れる。
- $i > j$の時、第（$N+j-i$）番目のチップの第jビット目にa_{ij}を入れる。

②の方式（図2の②）

第（$i \oplus j$）番目のチップの第jビット目に入れる。

ただし、

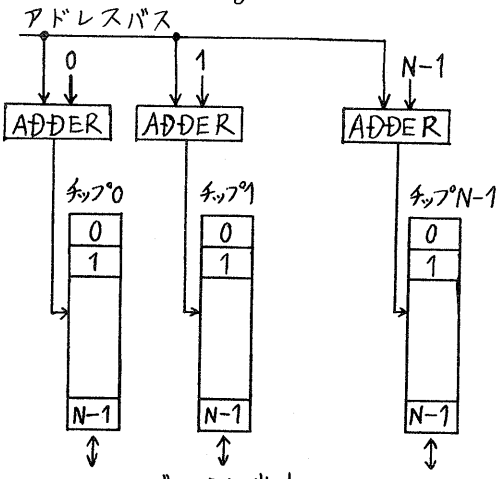
$$\begin{cases} i \oplus j = (i_0 \oplus j_0) + (i_1 \oplus j_1) \times 2^1 + \\ \qquad \text{---} + (i_{N-1} \oplus j_{N-1}) \times 2^{N-1} \\ i = i_0 + i_1 \times 2^1 + \text{----} + i_{N-1} \times 2^{N-1} \\ j = j_0 + j_1 \times 2^1 + \text{----} + j_{N-1} \times 2^{N-1} \end{cases}$$

⊕は排他的論理和である。

以上の様な配置を実現するアドレス変換回路は、図3のようになる。①は$\log_2 N$ビットの加算器をN個必要とするのに対し、②は$\log_2 N$個のEORゲートだけでよい。

図3．アドレス変換回路

① Skewed Array方式の場合

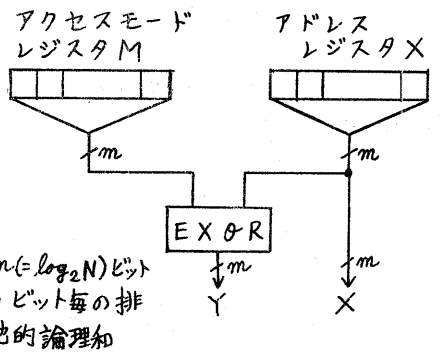


- ワードスライスアクセス

加算器の入力にワードアドレスをセット。

・ビットスライスアクセス
　加算器をバイパスし、ビット番号を直接各メモリチップのアドレス端子に接続。

②ＥＯＲ　Skewed Array方式の場合



チップ番号で（$i=0,1,\cdots,N-1$）のメモリーのアドレス端子$A_0A_1\cdots A_{m-1}$と上のX，Yバスの接続法は、

$$A_k=\begin{cases} i_k=0\text{の時} & X_k \\ i_k=1\text{の時} & Y_k \end{cases}$$

である。ただし、

$$\begin{cases} i=i_0+i_1\times 2^1+\cdots+i_{m-1}\times 2^{m-1} \\ X=X_0+X_1\times 2^1+\cdots+X_{m-1}\times 2^{m-1} \\ Y=Y_0+Y_1\times 2^1+\cdots+Y_{m-1}\times 2^{m-1} \end{cases}$$

・ワードスライスアクセス
　Mレジスタにall 1をセット。
　Xレジスタにワードアドレスをセット。
　M＝all 1より、Y＝～X。（図2の②の右のアドレスの与え方のようになる。）

・ビットスライスアクセス
　Mレジスタにall 0をセット。
　Xレジスタにビットアドレスをセット。
　M＝all 0より、Y＝X。

［データの並べ替え］

行、列方向のアクセスを共に可能にするような、データの配置をするためには、上記の様なアドレス変換回路だけではだめで、データの並べ替えの回路も必要である。これは、アドレスが異なる時、データの各ビットの配置を全て変えないと、同一チップに同じワード番号（ビット番号）のビットが入ってしまうからである。従って、データを読み出す時は、元の順序に戻すために、書いた時と逆の入れ替えを行う。データの読み書きの両方に対し、入れ替えが必要な事、アドレス毎に並べ替えが異なる事に注意が必要である。

具体的に、①、②の方式の場合の並べ替えの操作は次の様になる。

ｎ語のワードスライスアクセス、又は、ｎビットのビットスライスアクセスをする時、

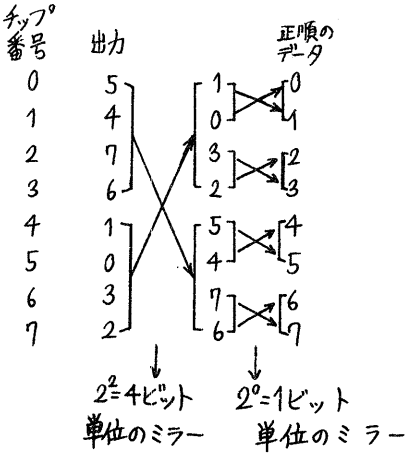
①の方式ーデータをｎビットサイクリックシフト。読み出し時には、書き込み時と逆方向にシフト。

②の方式ー$i=i_0+i_1\times 2^1+\cdots+i_{m-1}\times 2^{m-1}$とすると、$i_k=1$（$k=0,1,\cdots,m-1$）を満たす、全ての$k$に対し、$2^k$ビット単位の鏡像反転（ミラー）操作を順に行う。

②の方式をわかりやすくするため、例によって示したのが、図4である。

図4、ＥＯＲ　Skewed Array方式の場合のデータの並べ替え

例ー図2の②でｎ5ビットのビットスライス出力。　$5=1\times 2^2+0\times 2^1+1\times 2^0$



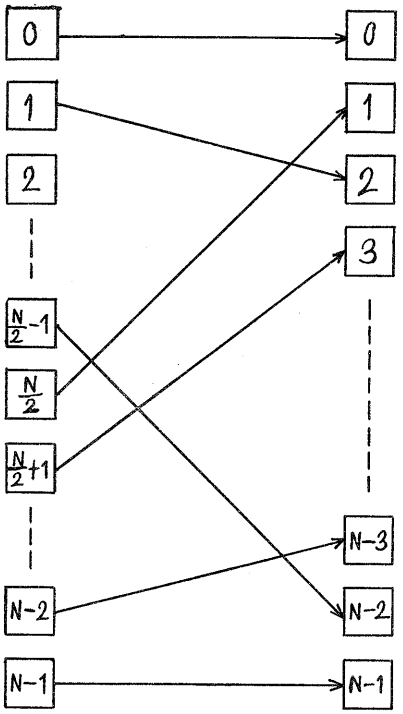
次に、このようなデータの並べ替えの操作を容易に実現する回路について述べる。

[Shuffle-Exchange 回路](4)(5)

先ず、Perfect Shuffle と呼ばれる回路について、説明する。図5に示したものがそれで、単なる結線回路に過ぎないが重要な性質を持つ。

図5. Perfect Shuffle

$$P(i)=\begin{cases}0\leqq i\leqq \frac{N}{2}-1 & 2i\\ \frac{N}{2}\leqq i\leqq N-1 & 2i+1-N\end{cases}$$



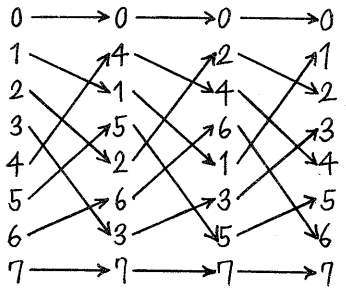
Perfect Shuffle の重要な性質は次の2点である。

1. $i=i_0+i_1\times 2^1+\cdots+i_{m-1}\times 2^{m-1}$ とすると、
$P(i)=i_{m-1}+i_0\times 2^1+i_1\times 2^2+\cdots+i_{m-2}\times 2^{m-1}$
となる。ただし、$m=\lceil \log_2 N\rceil$

2. $N=2^m$ の時、m回 Shuffle を繰り返すと、必ず元の順序に戻る。

1. の性質から、k回 Shuffle した後では、$P^{(k)}(i)$ のLSBが i_{m-k} になる。従って、この時の偶数番目と奇数番目のインデックスの隣り同士の差は 2^{m-k} である。つまり、m回 Shuffle をする途中で、最初に自分と 2^{m-1}, 2^{m-2}, ---, 2^1, 2^0 だけ離れていた点とこの順序で隣り同士になる。(図6の例参照)

図6. $N=8$ の時の $m=\log_2 N$ 回の Perfect Shuffle



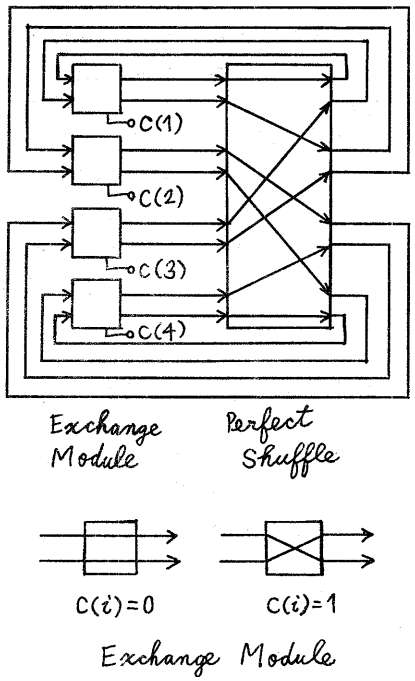
ステップ	1	2	3
隣同士のインデックスの差	4	2	1

この事から、各ステップで隣り同士の入れ替えを適当に行えば、データを0から $N-1$ 番目の任意の場所へ $\log_2 N$ 回で移動する事ができる。そこで、Perfect Shuffle をした後に、隣同士の入れ替えの回路を付加し、入れ替えの決定は、制御ビット $C(i)$ ($i=0,1,\cdots,\frac{N}{2}-1$) によって行えば、種々の入れ替え操作が可能である。この様な回路は、Shuffle-Exchange 回路と呼ばれ、その構成は図7に示した。

k回目の Shuffle 後の制御ビットを $C^{(k)}(i)$ ($k=0,1,\cdots,m-1$) とする。m回の Shuffle-Exchange によって、入れ替えを行う場合、すべての $C^{(k)}(i)$ の組合せによって、任意の入れ替えが可能となるわけではない。しかし、サイクリックシフトや、EO-R-Skew の並べ替え等の重要な入れ替えが可能である。

$\frac{1}{2}N\log_2 N$ 個の制御ビット $C^{(k)}(i)$ をどのように決めたらよいかについては、アルゴリズムが示されている。(6) 又、この制御ビットを発生するための回路も提案されている。(5) これは、制御ビットもデータと共に、Shuffle 回路を通

図7. Shuffle-Exchange回路



し、隣り同士のEXORをとって、新しい制御ビットとする方法である。しかし、EOR-Skew の並べ替えと2^pビット（P＝0，1，---，m－1）のシフトの2つの機能だけに限定すると、制御ビットを発生する回路は簡単になる。以下に、その制御ビットの与え方を示す。

(1) EOR－Skew の並べ替え

この場合は、各ステップ内で、すべての制御ビットに共通の値を与える。そして、各ステップでの制御ビットの値は、図4で示したように、ビット又はワードアドレスをmビットの2進数に展開して、MSBから順に1ビットずつ取り出してくればよい。これは、kステップ目で、制御ビットを1にすると、2^{m-k}ビット単位の鏡像反転になるからで、Perfect Shuffle の1、の性質に基づいている。EOR－Skew の並べ替えは、$\log_2 N = m$ステップで終る。

(2) 2^pビット単位のシフト

例として、N＝16，m＝4の場合に、2^0，2^1，2^2，2^3の各ビット数だけシフトする時の制御ビットパターンを図8に示した。

図8. 制御ビットのパターン（N＝16）

制御ビット	ステップk ①	②	③	④	⑤	⑥	⑦	グループ
C(0)	0	0	0	1	0	0	0	
C(1)	0	0	0	1	0	0	0	G(1)
C(2)	0	0	0	1	0	0	0	
C(3)	0	0	0	1	0	0	0	
C(4)	0	0	1	1	0	0	0	G(2)
C(5)	0	0	1	1	0	0	0	
C(6)	0	1	1	1	0	0	0	G(3)
C(7)	1	1	1	1	0	0	0	G(4)

シフト量	ステップシーケンス
2^0	①→②→③→④
2^1	②→③→④→⑤
2^2	③→④→⑤→⑥
2^3	④→⑤→⑥→⑦

[ビット]

シフト操作、特に2^pビットのシフト操作の時の、制御ビットのパターンは次の2つの性質を持つ。

1．k＝1，2，---，m－Pに対し、

$$C^{(k)}(i) = \begin{cases} 0 & 0 \leqq i \leqq \frac{N}{2} - 2^{p+k-1} - 1 \\ 1 & \frac{N}{2} - 2^{p+k-1} \leqq i \leqq \frac{N}{2} \end{cases}$$

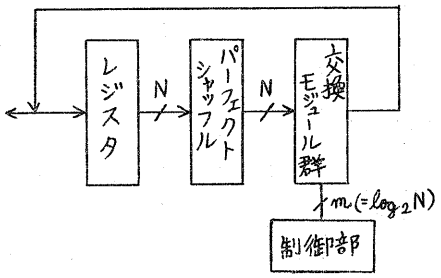
つまり、1の列がステップ毎に倍の長さになっていく。

2．第m－Pステップで制御ビットは全て1となり、P≠0ならば、残りの第（m－P＋1）から第mステップまで制御ビットは全て0となる。

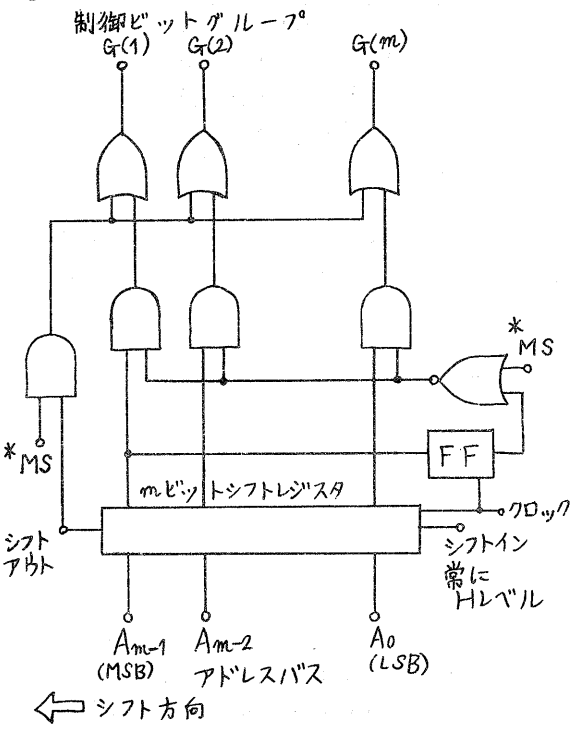
これらの性質を用いて、N/2個の制御ビットをm個のグループに分けて制御すればよい事がわかる。図8でわかるように、最初のステップの初期パターンが決まれば、後の変化の仕方は同じである。以上より、EOR－Skew の並べ替えと、2^pビット単位のシフトを$\log_2 N$ステップで行う回路は、図9のようになる。

図9．データの並べ替えとシフトの兼用回路（Shuffle-Exchange）

①全体の構成図



②制御部の回路



MS（モードセレクト）

- Hレベル----EOR-Skewの並べ替え
- Lレベル----2^pビット単位のシフト

アドレスバス

- EOR-Skewの並べ替えの時、ビットアドレス、ワードアドレスが入る。
- 2^pビットのシフトの時
  00----011---1 をセット。
  （m-pビット、pビット）

[3] 二次元記憶を用いた連想処理システム

[2]で述べた方法によって、二次元アクセス記憶が構成できる。この二次元記憶に、ビットスライス型のマイクロプロセッサを付加し、並列比較等の連想処理を行うシステムを図10に示す。このシステムでは、データを行単位（ワードスライス）で書き込み、列単位（ビットスライス）で読み出しながら、比較レジスタの対応するビットと比較する事により、データの並列探索を行う。

図10．連想処理システムのブロック図

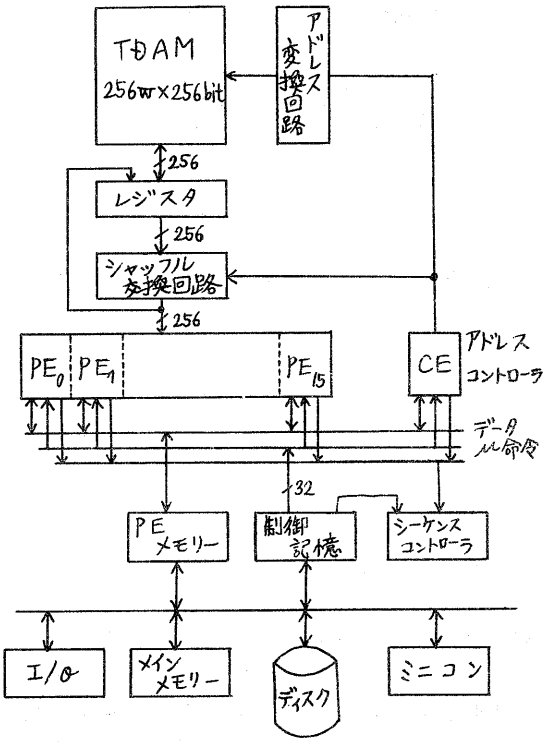
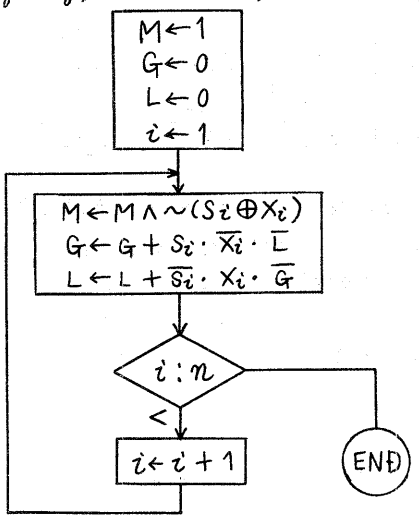


図11に、Equality, Greater Than, Less Than, Max, Min のサーチアルゴリズムを示す。図11のアルゴリズムは、簡単な論理式で表わされるので、各アルゴリズムをマクロ命令として定義し、各マクロはマイクロプログラムで記述する。個々の演算は、ビット直列に行うので、並列度は256をフィールド長で割った数になる。PEには、レジスタファイル、マスク機能、バレルシフタ等、多数の機能を持つマイ

クロプロセッサを使う事により、応用範囲が広がる。

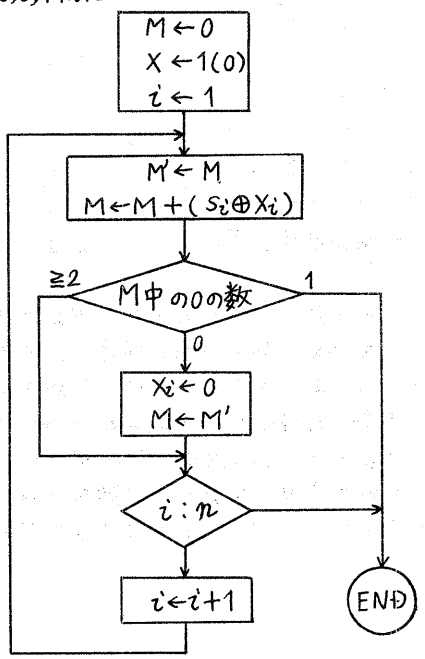
図11. サーチアルゴリズム

① Equality, Greater Than, Less Than



S_i データのi番目ビットスライス
X_i 比較レジスタのi番目ビット目
G Greater Thanの語を示す。
L Less Thanの語を示す。
M EQualの語を示す。
ただし、iはMSBから調べる。

② Max, Min



[4] 連想処理システムの応用

(1) パターン認識

パターン認識への応用を考えた場合、種々の認識アルゴリズムに対する性能比較のためのハードウェアシミュレータとして有用である。汎用計算機では処理時間がかかり、又、専用のシステムでは他の方式との比較が難しい。このシステムでは、各種のアルゴリズムをマイクロプログラムで記述するので、変更が容易であり、また、パターン認識用のマクロ機能の設計や評価がやり易い。

細線化、平滑化等の前処理の他に、特徴抽出の際に役立つ処理として、連結点の処理、ビットカウント、最左端のビットアドレス等がある。又、認識の最終段階における辞書との照合では、[3]で述べたアルゴリズムを用いて、並列サーチができる。前半の処理は、ほとんどがPEの機能によって実現可能である。当然PEのマイクロプロセッサでファームウェア的に処理しきれない部分や、効率の悪い処理については、外部に専用回路を付加する。

例として、100×100の2値画像の細線化、平滑化の処理時間を、PEのサイクルタイムを200nSと仮定して、評価した。(図12参照) これは、3×3の窓によるアルゴリズムであるが、9個の点をPE内のレジスタファイルに取り込み、1行又は1列の256画素分同時に並列処理する。従って、パターンの大きさが256×256を越えない時、処理時間は画素数の平方根に比例する。

図12. パターン認識への応用の例

① 2値画像の平滑化

アルゴリズム

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

I、x_1～x_9のうち、1が5個以下なら、x_5←0

II、x_1, x_3, x_7, x_9のうち、1が2個以上なら、x_5←1。

III、x_2, x_4, x_6, x_8のうち、1が2個以下なら、x_5←0。

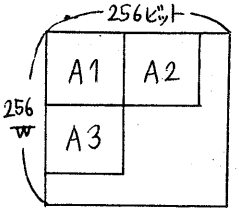
I, II, IIIの順に処理する。

処理時間

処理	プログラムステップ数	1行の処理時間[μS]	100×100の場合[mS]
I	113	22.6	2.26
II	49	9.8	0.98
III	50	10.0	1.00
計	212	42.4	4.24

ただし、ステップ数はマイクロプログラムをハンドアセンブルして数えた。

② 2値画像の細線化(Thinning)



A1：原画
(100×100)

I. $x_1 \sim x_9$ のうち1が5個以下なら、
$x_5 \leftarrow 0$。 結果はA2へ入る。
(境界除去)

II. 1. A3←A1∧~A2
(境界がA3に入る)

2. A2に対し、$x_1 \sim x_9$のうち、少なくとも1つが1なら、$x_5 \leftarrow 1$にして、結果をA2に入れる。(膨張)

3. A3←A2∧A3

(2と3は、線幅が1ないし2の線や正方形などが除去されるのを防ぐためにある。)

4. A3に対し、$x_1 \sim x_9$のうち、1が3個以上なら、$x_5 \leftarrow 0$にし、A3に格納。(連結性の維持)

5. A3がall 0なら終了。

6. A1←A1∧A3

7. ステップIへ戻る。

処理時間

処理	プログラムステップ数	1行の処理時間[μS]	100×100の場合[mS]
I.	115	23.0	2.3
II.1.	5	1.0	0.1
2,3	23	4.6	0.46
4,6	118	23.6	2.36
5	5	1.0	0.1
計	266	53.2	5.32

(注) この合計は、1回の細線化処理時間で、実際には、これのn倍(反復回数)である。データの線幅などによって、nは変わるが、n=10~20としても、100×100の2値画像は53.2~106.4mSで処理できる。

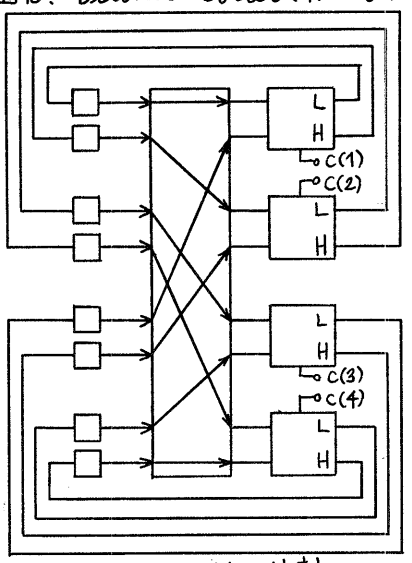
(2)データベースマシンへの応用

データ構造として、リレーショナルモデルを採用すれば、表に対する検索であるから、このシステムに適したモデルと言える。[3]で述べた連想処理アルゴリズムを用いて、リレーショナルモデルで用意している述語機能を効率良く実装することができる。

検索条件を満たす語は、PE内のレスポンスレジスタにその位置が示される。語数を調べるためには、このレジスタ中の1の数を数えるビットカウント処理が必要である。これは、各PE毎に16ビットずつまとめてシフトカウントし、次に16台のPEのカウントの和をとる。又、条件を満たす語の先頭アドレスを示すアドレスリゾルバーは、プライオリティー・エンコーダを何段か縦続接続する事によって、容易に作成できる。

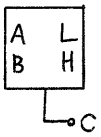
リレーショナル・アルジェブラにおけるJOINのような極めて効率の悪い操作も、ドメインについてSORTされていれば、効率が良くなる。Batcherの提案したBitonic Sorter[3]は、図13のように、Perfect Shuffleと比較エレメントから成り、入力データ数Nに対し、$(\log_2 N)^2$ステップでソートするネットワークである。

図13、Bitonic Sorter（N = 8）



レジスタ　パーフェクト　比較
シャッフル　エレメント

比較エレメント



- C = 0 の時
  - L = min（A，B）
  - H = max（A，B）
- C = 1 の時
  - L = max（A，B）
  - H = min（A，B）

比較エレメントはわずか13個の2入力NORで製作できるが、同じ処理をPEのマイクロプロセッサ内で行う事もできる。この場合、1ビットの処理に十数ステップのμ命令が必要である。従って、例えば、このシステムで、比較エレメントを付けずに、Perfect ShuffleとPEの組合せで、16ビット256 Wordのデータのソートをすると、処理時間は数mSecかかる。当然、専用の比較エレメントを付加すれば、速度は1桁から2桁早くなる。ゲート数も少ないし、交換モジュールと比較エレメントの機能を両方持つような回路を、Perfect Shuffleと組合せれば、EOR-Skewの入れ替え、2^pビットのシフト、Bitonic Sortの3つの機能を全て、一つの回路で実現できる。SORT機能の付加は、データベース管理への応用の場合は特に必要と思われる。

(3) その他の応用

Perfect Shuffleは、FFT（高速フーリエ変換）の際のShuffle操作にとって有用である。その他、高速乗算、多項式計算、偏微分方程式の解等、2次元アクセス記憶やPerfect Shuffleの特徴を活用できる応用分野は広い。しかし、本研究ではパターン認識とデータベース管理への応用を目標とする予定である。

[5] まとめ

以上で、2次元記憶を用いた連想処理システムが、比較的容易に構成できること、パターン認識やデータベース管理の分野で大きな処理能力を発揮しうる事を示した。2次元記憶を構成する上で、Perfect Shuffleの果たす役割は大きく、PE間のデータ交換にも有用である。さらに、回路的には1対1の接続であるから、ファンアウトが小さくて済み、速度の向上も期待できる。

他方、PEにマイクロプロセッサを用いる事により、

- 価格の低下
- 設計の容易さ
- 問題に適応した命令セットの設計、変更が容易—ファームウェア化
- 機能の拡張性

等の利点がある。

現在、本研究では、性能評価のためのシミュレーションや、問題への適応性のチェックを行っている。従来、大規模な連想プロセッサで実現された例は少なく、今後ここで述べたようなシステムが、製作の容易さという点で有利と思われる。

[参考文献]

1、K.J. Thurber
「Associative and Parallel Processors」
Computing Surveys, Vol.7, No.4,
pp.215-255, 1975

2. K.E. Batcher
「STARAN parallel processor system hardware」
NCC pp. 405-410, 1974

3. K.E. Batcher
「Sorting networks and their applications」
SJCC pp. 307-314, 1968

4. H.S. Stone
「Parallel processing with the perfect shuffle」
IEEE C-20, pp. 153-161, 1971

5. T. Lang and H.S. Stone
「Shuffle-exchange network with simplified control」
IEEE C-25, pp. 55-65, 1976

6. D.H. Lawrie
「Access and alignment of data in an array processor」
IEEE C-24, pp. 1145-1155, 1975

7. D.E. Knuth
「The Art of Computer Programing」
vol. 3, Addison-Wesley, 1969

8. 元岡、田中、上森
「パターン処理用プロセッサのアーキテクチャ」 昭51. 情報処理全国大会 19.