計算機アーキテクチャ　80−12
（１９９０．１．２６）

ＣＡＭを用いた機能メモリ型並列プロセサＦＭＰＰ

辻本　泰造　　安浦　寛人　　田丸　啓吉
京都大学工学部

機能メモリの各語を一つのプロセサと見なすことで，メモリ集積回路技術の進歩をそのまま大規模な超並列計算機構の実現に利用することができる．われわれは，このような並列アーキテクチャを機能メモリ型並列プロセサアーキテクチャＦＭＰＰと呼ぶ．本報告では，代表的な機能メモリであるＣＡＭ（Contents Addressable Memory）を基本にしたＦＭＰＰアーキテクチャを提案する．メモリの各語は１ビットのプロセサとして働き，システム全体では語数分の並列度を持ったＳＩＭＤマシンとなる．ここでは，このアーキテクチャの上での主な基本演算の計算時間の評価を行なうとともに，応用例を示す．現在の集積回路技術で，数百万プロセサが実現できると考えられる．

Functional Memory type Parallel Processor Architecture Based on CAM: FMPP

Taizo Tsujimoto, Hiroto Yasuura and Keikichi Tamaru

Department of Electronics, Faculty of Engineering
Kyoto University, Kyoto, 606 JAPAN

We have been investigating a parallel processor architecture using functional memories, called FMPP (Functional Memory type Parallel Processor architecture). In this report, we show an FMPP architecture based on CAM (Contents Addressable Memory). Each word of the memory is a processor unit performing a bit serial computation. The architecture is a kind of SIMD, and the number of processors can be increased rapidly according the progress of memory LSI technology. More than millions of processors will be implemented by the recent technology. We show application fields of the FMPP architecture and evaluate algorithms for basic operations on the architecture.

1．まえがき

集積回路技術の急速な進歩により、高い並列性を持つハードウェアが実現できるようになり、従来の逐次型計算機では実用的に解けなかったような問題を解くための専用ハードウェアの設計が盛んになってきた。並列度の高い計算機アーキテクチャにおいては、各プロセサの複雑さと同様にプロセサ間の通信機構の複雑さがハードウェアの規模を規定する。また、プロセサ間の通信やプロセサへのタスクの割当のスケジューリングなどが計算時間の大きなオーバーヘッドとなる。

集積回路において、最も集積化の増大が速いのはメモリ回路であり、３年で４倍の向上がしばらく続くとみられている。これに対してランダムロジックや複雑な通信機構を含む回路は設計の困難さなどが主な理由となってメモリ回路のような集積度の向上は得られていない。並列計算機のなかでも、構成単位のプロセサが１つの汎用計算機に匹敵するような高度なものであったり、プロセサ間の通信網が複雑なものは、同じ理由から集積回路化・集積度向上が未だ困難なことが多く、実用に十分な並列度を得ることは難しい。１次元や２次元のアレイプロセサあるいはシストリックアレイなどは、おなじ回路の繰り返しによる設計の容易さが特徴の１つであり、プロセサ間通信はバス通信のような平板なものかあるいはなるべく近傍のもの同士だけにして通信機構が複雑になるのを避けているので、最も集積化に適した並列計算機構と言われている。それでも単位プロセサの機能をある程度もたそうとすると、メモリ回路のような集積度を得ることはできなくなる。

メモリの１ビットあるいは１ワードといった小単位ごとに簡単な演算機能を付加したアーキテクチャは、論理付きメモリや機能メモリと呼ばれ、応用指向のメモリとして種々のものが提案されている[2]。その構成単位である、何ビットかのメモリと演算回路からなる部分を、それぞれ並列に動作させた場合、この構成単位を１つのプロセサとしてメモリアレイ全体でプロセサアレイを形成すると見なすことができる。このようなメモリに非常に近い構造の回路はメモリ集積技術が応用しやすく、メモリ同様な高集積度・集積度向上が見込まれる。

この様に、非常に単純な構造をとることによって現在のメモリＶＬＳＩ技術の発展を享受し、１チップあるいは数チップに十分な数のプロセサを盛り込む事をめざしたアレイプロセサを、我々は機能メモリ型並列プロセサ（Functional Memory type Parallel Processor：ＦＭＰＰ）と呼ぶ。[19] ＦＭＰＰは多くのアレイプロセサ同様ＳＩＭＤ（Single Instruction stream Multi Data stream）型並列計算機構である。プロセサ間の通信は、メモリのビット線に対応するバスによる通信や、隣接ワード間程度の近傍通信のみを考える。

連想メモリ（Associative Memory または Contents Addressable Memory：ＣＡＭ）は従来からよく知られている機能メモリの一種であり、記憶内容によるアドレシングが可能なものである。そのハードウェアの提案は１９５０年代にまでさかのぼり、それ以来様々なＣＡＭハードウェアとそれを用いた計算機システムが提案されてきている[1],[3]~[9]。
その中には、連想プロセサと呼ばれる、ＣＡＭをプロセサアレイ的に用いる、すなわちＦＭＰＰと同様な考え方に基ずくものも多く含まれている。例えば Goodyear Aerospace のＳＴＡＲＡＮや Sanders Associates のＯＭＥＮ－６０シリーズなどが’７０年代に商用化されている[10][11]。
しかし十分な性能を得るためには装置規模は大きくならざるを得ず、応用範囲の特殊性も加わって、広く使用されるには至らなかった。最近になってかなりのビット数をワンチップに集積したＣＡＭが開発されるようになり[12][13]、ハードウェア量での制限が緩くなったので、ＣＡＭの新たな応用を考える研究が再び盛んになってきている。最近の例では NTT のLISP/Prologマシン[15]、MITのﾃﾞｰﾀﾍﾞｰｽｱｸｾﾗﾚｰﾀ、[16]早大の図形・配線処理用マシン[17]、NECの文字列検索マシン、京大のUNION

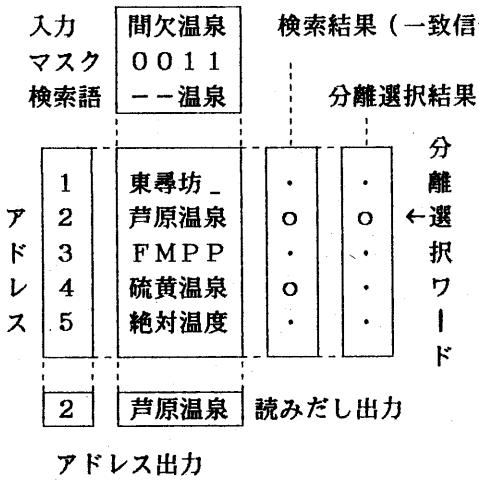


図１　連想メモリ動作概念図

-FINDメモリなどがある。[18]

我々はＦＭＰＰの典型的なものとしてＣＡＭを用いたものについて研究を行なっている。これまでに、ある種の組合せ／最適化問題に適用可能であることを示した。[19]
本論文ではＣＡＭの基本的なアーキテクチャについて紹介し、それをＦＭＰＰとして用いる場合の変更部分やアルゴリズムについて検討する。そして応用例として並列に表計算を行なう方法を示す。

２．ＣＡＭの基本回路

連想機能の概念を説明する。通常のメモリにおいては、アドレスによってアクセスデータが指定される。連想メモリでは検索データが与えられ、これと各ワードの記憶データが同時に並列に比較され、一致した（あるいは、指定された大小関係を満たす、互いに一定のハミング距離内にある、などある種の「関係」が成り立つ）ワードにおいて一致検出を示すフラグがたてられる。これを選択ワードという。検索データはビット単位やバイト単位でマスクできるようになっている。選択ワードに対してアクセスが行なわれる。読み出しは１つのワードしか行えないので、複数の選択ワードから複数選択分離回路によってただ１つ分離選択ワードを指定して行なわれる。（図１）

２．１回路構成

ＣＡＭの基本的な回路構成は、入出力ポート、データレジスタ（dbit）、マスクレジスタ（MR ,dbit）、メモリアレイ、アドレスデコーダ／エンコーダ、ワード処理回路（一致検出フラグを含む。演算機能や近傍との通信機能をもたせる事が多い）、複数選択分離回路（Multiple response resolver）の各ブロックから構成される（図２）。入出力ポートはアドレスバス（入出力）、データバス（入出力）、命令入力（Command input：入力）、選択プロセサ検出線（Detection line：出力）の４種である。メモリアレイはｗワードの１次元配列であり、ｄ×ｗ個の連想メモリセルの配列で構成される。検索においてマスクレ

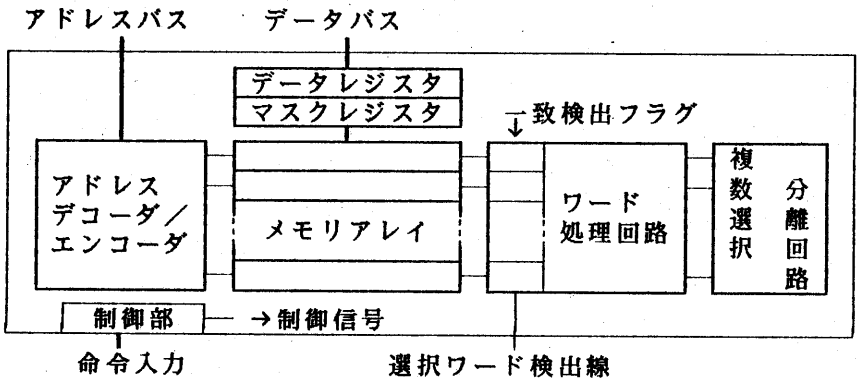
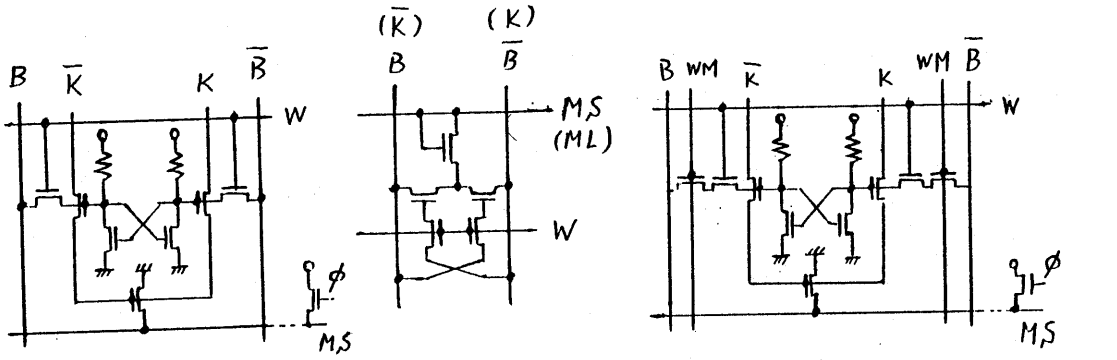


図２　　ＣＡＭの基本回路構成



（ａ）ＳＲＡＭ型　　（ｂ）ＤＲＡＭ型　　（ｃ）ＳＲＡＭ型
（書き込みマスク可）

Ｂ／$\overline{B}$：ビット線　　Ｋ／$\overline{K}$：検索データ線
Ｗ：ワード線　　ＭＳ：一致検索線　　ＷＭ：書き込みマスク線

図３　連想メモリセルの構造

ジスタによる１bit単位でのマスク機能を持つとする。

ＣＡＭの連想機能（並列比較）を実現するためのメモリビットセルの回路を図３（ａ，ｂ）に示す。（ａ）はＳＲＡＭによる[14]、（ｂ）はＤＲＡＭによる実現[20]である。通常のＲＡＭセルに比較回路が付加されている。（ｂ）はＤＲＡＭセル２つを巧妙に用いることによって「０」「１」以外に「don't care」も記憶することができる。並列書き込みにおいて１bit単位でのマスク機能を実現するには、各ビットにおいてマスク情報を放送するビット線方向の通信線を設け、これによって制御されるトランジスタ数個をビットセルの書き込み用のトランジスタに付加する必要がある。これをＳＲＡＭで実現した場合図３（ｃ）の様になる。今後はこの回路を採用する。

２．２基本動作

ＣＡＭの基本的な動作として以下のようなものを考える。

１）一致検索：REF operand

値 operand に MR によるマスクをほどこして検索語を生成し、各ワードにおいてそれぞれその内容とマスク付き比較を行う。一致信号MSは一致したとき１、そうでないとき０となる。

２）選択ワード読みだし：READ

選択ワードのアドレス／データをバス出力に読みだす。選択ワードが複数ある時は複数選択分離回路によってその中の一つが選ばれる。選択ワードがないときはこの命令は無効。

３）選択ワード並列書き込み：

WRITE operand

値 operand に MR によるマスクを行い選択ワードに並列に書き込む。

４）マスクレジスタへの書き込み

：MASKSET operand

値 operand を MR へ書き込む。

５）リセット：RESET operand

operand で指定される範囲をリセット。指定範囲としては全メモリ、全レジスタの初期化または一致検出フラグなどのワード処理回路のレジスタなどが考えられる。

この他に通常のメモリと同様のアドレス指定による読みだし／書き込みも変更を加えずに実現できる。

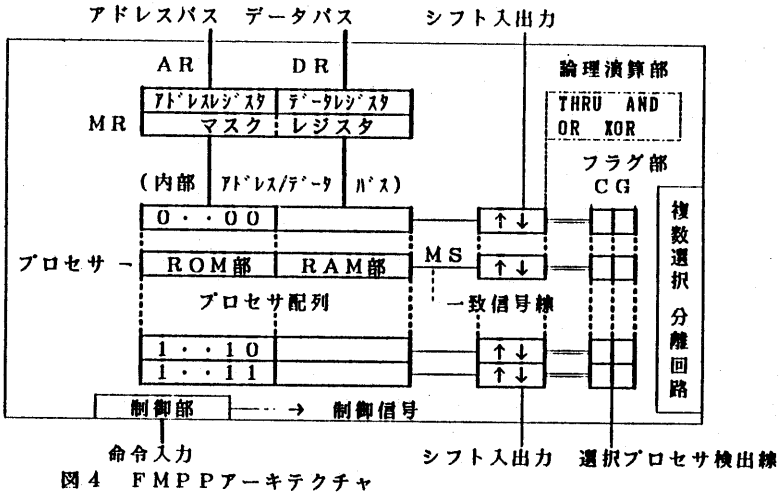


図４　ＦＭＰＰアーキテクチャ

３．ＣＡＭを用いたＦＭＰＰ

前章でのべたＣＡＭをＦＭＰＰとして用いてるためにワード処理回路の構造を定義しておかなければならない。この部分の性能はＦＭＰＰの能力に直接反映するものであるが、メモリ構造を壊さずに設計するためには機能を非常に簡単なものに限定して注意深く選ぶ必要がある。ここでは簡単な論理演算部と２ビットのフラグレジスタから成るものとする。フラグはＣフラグ、Ｇフラグと呼ばれ、メモリアレイからの一致信号が論理演算部によってＣフラグに累算され、Ｇフラグはワードを不要化するために用いるものとする。メモリの１ワードと論理演算部、フラグによって１プロセサが構成されると見なし、以後「ワード」を「プロセサ」に言い替える。この回路は現在実現されているNTTのＣＡＭ[13]に主に次の３点の変更を加えたものとなる。（図４）[19]

（１）論理演算部にXOR機能を持つこと。

（２）通常のアドレスデコーダ／エンコーダを用いる代わりにメモリアレイの一部のビットフィールドをROM化してアドレスをもたせるようにしたこと。デコード／エンコード機能は失わずにさらにアドレスにおいてもマスク付き一致検索ができるようになる（一部にこの機能を持つＣＡＭは試作されている）。メモリアレイはROM部とRAM部に分割される。

（３）隣接プロセサ（アドレスが連続しているプロセサ）間でのＣフラグのシフト転送機能をこれまでの単方向から双方向に拡張すること。シフトのみの動作も可能とする。

３．１命令セット

本アーキテクチャは一種のＳＩＭＤマシンであり外部から与える基本命令は特殊な場合を除いて等しく全プロセサで並列に動作する。２．２で述べたＣＡＭの基本動作をもとにして以下のようなＦＭＰＰの基本命令を定義する。入出力データを示すoperandはaddress,dataに分けて記述される。

１）一致検索：REF function <address,data>

address,dataにMRによるマスクをほどこして検索語を生成し、各プロセサにおいてそれぞれそのROM部およびRAM部の内容とマスク付き比較を行う。一致信号MSの値と選択フラグＣの内容にfunctionで指定される論理演算（THRU/AND/OR/XOR、THRUはMSの値をそのまま与える）をほどこした値が新しいＣの内容になる。

２）選択フラグの隣接シフト：

SHIFT C direction

Ｃフラグの内容を隣接プロセサ間でシフト転送する。（Ｃi→Ｃi±1）

direction は転送方向（UPPER,+ ／ LOWER,-）。

３）シフト付き一致検索：

SHIFT direction REF function <address,data>

１）と２）の複合命令。機能的には、まず２）を実行してから１）を実行することと等価。

演算数	外部値		内部値	
演算	命令数	オーダー	命令数	オーダー
加算	5n+4	O(n)	9n+4	O(n)
転送 ※	(2)	(O(1))	7n	O(n)
乗算	$5n^2+4n$	$O(n^2)$	$9n^2+3n+4$	$O(n^2)$
*「縦」乗算	***	O(n)	***	O(n)
論理和／積 ※	2	O(1)	3n	O(n)
等価比較	1	O(1)	3n+1	O(n)
大小比較	2n+1	O(n)	5n+1	O(n)
最大値検索	---	---	2n	O(n)

※転送は転送元フィールド、転送先フィールド（各nbit）を指定し、1プロセサ内または隣接プロセサ間で転送コピーを行なう。（外部値の転送はただの書き込みになる）

論理積／和はビットごとに行なうものとする。

表1　基本演算手続きの命令数

（Ci ◎ MSi±1 → Ci±1 ，◎は演算function）なおfunctionとしてTHRUは指定できない。

4）選択プロセサ読みだし：READ C

選択プロセサ（C＝1，G＝0）のROM部／RAM部をアドレス／データバスに読みだす。選択プロセサが複数ある時は複数選択分離回路によってその中の一つが選ばれる。選択プロセサがないときはこの命令は無効。読み出されたプロセサはCフラグをリセットする。

5）選択プロセサ並列書き込み：

WRITE C <,data>

値dataにMRによるマスクを行い選択プロセサに並列に書き込む。

6）選択プロセサの不要化：GARBAGE

選択プロセサを不要化（G←1）する。

7）マスクレジスタMRへの書き込み

：MASKSET <address,data>

値address,dataをMRへ書き込む。

8）リセット：RESET region

regionで指定される範囲をリセットする。regionはall（全RAM部と全フラグ）、C（全Cフラグ）、G（全Gフラグ）の3通り。

これらの基本命令は現在のCAM[13]の命令とほぼ同等で、1サイクル100ns～200nsで実現可能と考えられる。

4．基本計算

3．で示した基本命令により、四則演算や比較などの基本演算手続きを記述することができる。数値（2進固定小数点数）のビット数をnとすると、加減算や大小比較はビットシリアルに行なわれるので計算時間はO（n）となり、乗除算はさらにビットごとの加減算に分解されるのでO（n^2）となる。記述例として加算／累加／乗算のアルゴリズムを図5に示す。またいくつかの基本演算手続きの命令数を表1に示す。なお、数値をワード線沿いではなくビット線沿いにもたせる手法が考えられる[21]。この場合各ビットにワード処理回路（1bitプロセサ）が割り当てられるのでビットパラレルに計算でき、累加や乗算などはキャリーセーブアダーのようなアルゴリズムで計算量のオーダーが1つ小さくなる。ただし、本質的に伝搬の性質を持つ計算には効果が無い。また、この方法ではこのままのハードウェアならば外部とのアクセスが並列に行えなくなるという欠点があり、それが可能になるようにするにはハードウェアが複雑になる危険があるのでよく検討する必要がある。

5．性能評価－関数計算の並列処理

計算機において、ある関数fのある引数xに対する関数値f（x）を求める必要が生じる場合、必要な時点で直接f（x）が計算されるという方法と、予めメモリ上に考えられるすべてのxに対するf（x）の表をもっておくという方法がある。両者の中間的な方法（たとえば、粗い関数表をもっておいて間を簡単な計算で補間する）がとられる場合もある。表をもっておく方法は

```
/*/*    ADDITION  (b←b+a)     */*/
for i=1 to m {  /* ビット直列加算ループ */
                /*  ↓a_i  b_i  c */
        MASKSET  <,0~010~010~010~0>
        REF THRU <,---0---0---1--->
        WRITE C  <,---0---1---0--->
        REF THRU <,---0---1---1--->
        WRITE C  <,---0---0---1--->
        REF THRU <,---1---0---0--->
        WRITE C  <,---1---1---0--->
        REF THRU <,---1---1---0--->
        WRITE C  <,---1---0---1--->
}
```

（a）加算アルゴリズム

```
/*/*    MULTI    (x←a*b)     */*/
MASKSET   <,0~0>   *1
REF THRU  <,--->   *1
        /*    c   ↓xの全ビット */
MASKSET  <,0~010~01~10~0>   *2
WRITE    <,---0---0~0--->   *2
for i=1 to m {  *3
    for j=1 to m {  *4              *5
                  /* ↓a_i  b_j  c   x_{i+j-1}*/
        MASKSET  <,0~010~010~010~010~0>
        REF THRU <,---1---0---1---0--->
        WRITE C  <,---1---0---0---1--->
        REF THRU <,---1---0---1---1--->
        WRITE C  <,---1---1---0---0--->
        REF THRU <,---1---1---0---1--->
        WRITE C  <,---1---1---1---0--->
    }
                /*                  ↓x_{i+m} */
    MASKSET      <,0~010~000~010~010~0>   *6
    REF THRU     <,---1-------1---0--->   *6
    WRITE C      <,---1-------0---1--->   *6
}
```

＜備考＞
＊1：全セル選択
＊2：桁上げビットと乗算フィールドのクリア
＊3：aのビット分解ループ，a_i=1のプロセサにおいてx ← x + b<<(i-1)
＊4：ビット直列加算ループ
＊5：乗算フィールドの加算位置指定がbのシフトを実現している
＊6：桁上げビットの格納とクリア

（b）（内部値）乗算（累加）アルゴリズム

図5　FMPP基本演算アルゴリズム例（加算／累加／乗算）

f（x）の見かけの（利用時の）計算時間がメモリアクセス時間だけですむが、前処理として表を生成するのに非常に時間がかかる。したがってこの方法は、一般によく使われる関数に対して、予め外部記憶装置やROMに関数表が準備されていて必要なプログラムにおいてアクセスされるという場合がほとんどである。したがって、例えばプログラム特有の関数を計算したい場合に、プログラムごとに前処理として関数表を計算することは非常に時間がかかり現実的でない。そこで、FMPPを用いて関数表を各関数値について並列に計算し保持しておく事ができれば、前処理の時間を縮小することができ、関数fが頻繁に変更されるような応用も現実的になる。

関数fを計算する3つの方法（直接計算、関数表の逐次計算による生成、関数表のFMPP上の並列計算による生成）の累積計算時間を比較すると図6のようになる。累積計算時間は、［前処理（表計算）時間］＋［関数値利用回数］×［（見かけの）計算時間］となる。グラフの交点は［前処理時間の差］／［見かけの計算時間の差］

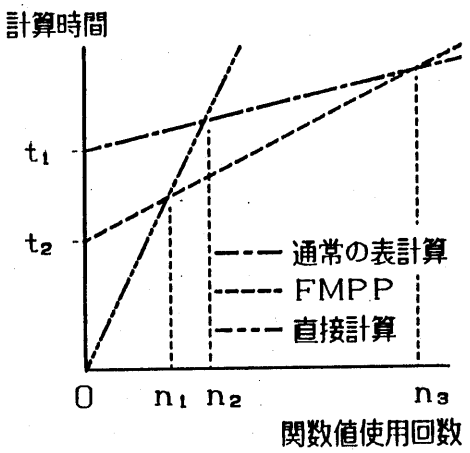


図6　関数計算コスト

となる。この様に中程度の計算回数ではＦＭＰＰが最も時間的に有利である。

６．おわりに

機能メモリ型並列プロセサＦＭＰＰはメモリの集積度の向上に追随できる並列計算機構である。今回はＣＡＭを用いたＦＭＰＰアーキテクチャとその応用例として表計算に付いて述べた。今後もさらに各種の応用に対応したＦＭＰＰが提案されるであろう。

＜参考文献＞

[1] T.Kohonen：" Content-Addressable Memories" Second Edition, Springer Series in Information Sciences 1, Springer-Verlag Berlin Heidelberg (1987).

[2] 古谷立美：" 応用指向メモリ"，情報処理，Vol.27, No.6, pp.601-606 (1986).

[3] C.Y.Lee and M.C.Paull：" A Content Addressable Distributed Logic Memory with Applications to Information Retrieval"，Proc. IEEE 51, pp.924-932 (1963)

[4] J.D.Barnard, F.A.Behnke, et al.：" Structure of a Cryogenic Associative Processor"，Proc. IEEE 52, pp.1182-1190 (1964)

[5] B.T.McKeever：" The Associative Memory Structure"，Proc. of the Fall Joint Comp. Conf.，pp.371-388 (1965)

[6] J.N.Sturman：" An Iteratively Structured General-Purpose Digital Computer"，IEEE Trans. C-17, pp.2-9 (1968)

[7] J.N.Sturman：" Asynchronous Operation of an Iteratively Structured General-Purpose Digital Computer"，IEEE Trans. C-17, pp.10-17 (1968)

[8] S.S.Yau and H.S.Fung：" Associative Processor Architecture-A Survey"，Comput. Surv. 9, pp.3-27 (1977)

[9] 市川，坂村，諸限，相磯：" 連想プロセッサＡＲＥＳ"，信学論（Ｄ），J61-D, 10, pp.743-750 (1978)

[10] R.W.Hockney and C.R.Jesshope （奥川，黒住 共訳）：" 並列計算機"，共立出版 (1984)

[11] P.H.Enslow,Jr. 編，村岡洋一 訳：" マルチ・プロセッサと並列処理"，近代科学社 (1976)

[12] 小倉，山田（慎）：" 連想メモリＬＳＩの現状と今後"，電子通信学会誌，Vol.69, No.7, pp.745-751 (1986)

[13] 小倉，山田（慎），山田（順）：" ２０Ｋｂ　ＣＭＯＳ　連想メモリＬＳＩ"，昭６１信学総全大，477 (1986)

[14] T.Ogura, S.Yamada, and J.Yamada：" A 20Kb CMOS Associative Memory LSI for Artificial Intelligence Machines"，IEEE Proc. ICCD'86, pp.574-577 (1986)

[15] 長沼，小倉，山田：" 連想メモリを用いたPrologマシンの構成と処理アルゴリズム"，情報処理学会記号処理研究会，32-3 (1985)

[16] J.P.Wade and C.G.Sodini：" A Ternary Content Addressable Search Engine"，IEEE Journal of Solid-State Circuits, Vol.24, No.4 (1989)

[17] 井出，石和，小島，鈴木，大附：" 連想メモリを用いた図形処理装置の試作"，信学技報，CAS87-106 (1987)

[18] 大久保，安浦，高木，矢島：" 連想メモリを利用したハードウェア向き単一化システム"，情処論文誌，Vol.28, No.9 (1987)

[19] 安浦，辻本，田丸：" 組合せ問題に対する機能メモリ型並列プロセッサアーキテクチャ"，信学論（Ａ），J72-A, 2, pp.222-230 (1989)

[20] J.P.Wade and C.G.Sodini：" Dynamic Cross-Coupled Bitline Content Addressable Memory Cell for High Density Arreys"，IEDM 85Tech. Dig., pp.284-287, Dec. (1985)

[21] B.A.Crane and J.A.Githens：" Bulk Processing in Distributed Logic Memory"，IEEE Trans. EC-14, pp.186-196 (1965)