EC78－2

パイプライン式SAMD計算機とその漢字認識への応用

PIPELINE SAMD MACHINE AND ITS APPLICATION TO THE RECOGNITION OF PRINTED CHINESE CHARACTERS

鈴木達郎　田中英彦　元岡達
Tatsuo SUZUKI　Hidehiko TANAKA　Tohru MOTO-OKA
東京大学　工学部
Faculty of Engineering, University of Tokyo

§0 序論

電子工学の発展により、回路のLSI化が進み、1チップのマイクロコンピュータを可能にし、低価格をもたらした現在は、アーキテクチャーの転換期であるとも言われるが、具体的に期待されるものの1つに並列処理がある。

一方これまで計算機が苦手としてきたパターン認識や人工知能の分野は本質的に大きな並列性を含んでいる。これは並列処理にとって絶好の応用分野である。

本研究はマイクロコンピュータを用いて大規模な並列度を実現するシステムを、パターン認識、特に印刷漢字認識システムをその具体的応用として、ハードウェア、ソフトウェアの両面から実現性を検討したものである。

内容は次の3本柱から成っている。

(1) パイプラインSAMD処理方式（ソフト）
(2) TOPSTAR（パイプラインSAMD処理用アーキテクチャ）の設計（ハード）
(3) 印刷漢字認識システム（アプリケーション）

(1)は大規模な並列度を達成し、それを制御できる処理方式の提案、(2)はそれを実行できる並列アーキテクチャの設計、(3)は応用として、印刷漢字を認識するシステムの作成である。

これらのうらづけとして(1)に関してはシミュレーションや理論的考察、(2)に関しては実際にパイロットモデルを試作して、設計通り動く事を確認、(3)に関しては数千回の認識実験を行った。

全体の構成は、§1でこの研究のねらいや考え方について述べ、上記の3本柱を§2～§4に、そして§5でこれら3つを統合し、その評価を§6にまとめるという形になっている。

§1 並列処理について

§1－1 並列処理の目的と問題点

並列処理、マルチプロセッサシステムの目的は次の3つの性質の向上を目指すことにまとめることができる。

(1) 信頼性
(2) 処理速度
(3) 価格性能比

一方並列処理の問題点は次のようである。

(i) ソフトウェアの問題点
　(1) 並列アルゴリズムの不足
　(2) 限られた応用分野
　(3) 高並列システムの制御OSの不足
(ii) ハードウェアの問題点
　(1) 価格
　(2) 結合方式
　(3) 制御方式

§1－2 パターン認識，人工知能などにおける並列性

脳の行なう情報処理の代表的なものであるパターン認識においてどのような並列性があるか考察してみる。

(1) 空間並列性

パターン認識の対象は一般に2次元空間内に存在するが、その処理は空間的広がりに対して、本来同時に（並列に）行なうべきものである。

(2) 認識(記憶との演算)の並列性

パターン認識の最終的な部分の「認識」は脳の中の過去の記憶と現入力パターンとの間に何らかの照合演算を行なっていると思われるが、それをシリアルに行なっているとは考えにくい。

例えば「連想」は処理時間のオーダーが記憶の量に比例しないので本質的に並列処理である。

(3) パターン分類の並列性

分類を二進トリー式にYes, Noのシリアルなくりかえしで行なうのは、誤差の伝搬の上からも、分類基準の強制的な順序づけを伏なうことからも好ましくない。

(4) 時間並列性

パターン認識などの動作は次々と来るデータに対して同じ動作をくりかえすことが多いが、これを時間とずらして空間的な並列性に変換することが可能である。このことを時間並列性があると呼ぶことにする。

• この研究ではパターン認識や人工知能などにおける並列性をうまく活かして、できるだけ、並列度の高い処理方式とそれを実現するアーキテクチャとを設計することを目標とする。

<CS(Critical Section)問題と並列度との関係>

並列処理で共通のデータに対して協力して演算を行なっていく場合、互いに処理に矛盾を生じないために、相互排除(Mutual Exclusion)が必要になるが、この相互排除の必要な範囲をそのプログラムのCSという。このCSの大きさは並列処理の並列度の上限を定める1つの要因となる。本研究ではできるだけ高い並列度を目指しているので、この上限を計算してみる。

CSを通過する平均時間 --- a sec
CSに入る頻度の平均 --- b^{-1} sec^{-1}

とすると、$\gamma = ab^{-1}$ がCSに入っている確率になる。

プロセッサの台数をn台とし、$k = 1-(1-\gamma)^n$ をどれか1つがCS内に入る確率とおくと、平均並列度は

$$\sum_{i=0}^{n-1}(n-i)\,k\,{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i} + n(1-\gamma)^n$$
$$= nk\sum_{i=0}^{n-1}{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i} - k\sum_{i=0}^{n-1} i\,{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i} + n(1-\gamma)^n$$
$$= nk + n(1-\gamma)^n - k\sum_{i=0}^{n-1} i\,{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i}$$
$$= n - k\sum_{i=0}^{n-1} i\,{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i}$$

ここで $\epsilon = k\sum_{i=0}^{n-1} i\,{}_{n-1}C_i\,\gamma^i(1-\gamma)^{n-1-i}$ がプロセッサを無駄に費やす割合ということになる。

$\gamma \ll 1$ としてϵを計算すると、

$$\epsilon = k\sum_{i=0}^{n-1} i\,{}_{n-1}C_i\,\gamma^i$$
$$= k\gamma\sum_{i=0}^{n-1}{}_{n-1}C_i\,\frac{d\gamma^i}{d\gamma}$$
$$= k\gamma\frac{d}{d\gamma}(1+\gamma)^{n-1}$$
$$= (n-1)k\gamma(1+\gamma)^{n-2}$$
$$\fallingdotseq (n-1)n\gamma^2(1+\gamma)^{n-2}$$
$$\fallingdotseq n(n-1)\gamma^2$$

となる。

従って $\gamma \sim n^{-1.5}$ となるときを上限と仮定して具体的な数値を代入すると、

$\gamma = 0.1$	⟶	$n = 4.64$
0.01	⟶	21.54
0.001	⟶	100.00
0.0001	⟶	464.16

となり、大きな並列度を目指す場合、実際に、影響があることがわかる。

§2. パイプラインSAMD処理方式

この章では、並列度の高いシステムを実現するための処理方式とその制御方式とを提案する。また、その方式が従来の計算機の苦手とするパターン認識などの分野に適していることを示す。

§2-1 SAMD並列処理とは。

Flynnによる、並列処理の命令とデータの流れに注目した分類(SISD, SIMD, MISD, MIMD)との関係において、SAMD(Single Algorithm-stream Multiple Data-stream)を次のように定義する。

[定義] SAMDとは

全体はいくつかの一連のアルゴリズムから成り、各アルゴリズムは複数のデータの流れを持つ。その並列度は、アルゴリズムごとに異なってもよい。

SAMDはFlynnの分類ではMIMDに含まれるが、これをSIMDの拡張と考えることもできる。即ちSIMDにおいて、そのプログラムカウンタが独立なもので、アルゴリズム(プログラム)は共通だが、各々その別々の場所を実行しているわけである。

データによる処理が完全に同じ場合は、SAMDはSIMDと同じ動作をするが、そうでない場合、例えば条件分岐などデータに依存した処理を行なう場合はSAMDの方が有利である。

§2-2 SAMD方式のパイプライン化

一般にマルチプロセッサの大きな問題は、いかに並列度を十分活かして処理を行なうことができるかであるが、このSAMD方式は各々のアルゴリズムの持つ並列性はもとより、以下に述べるようにパイプライン化することでさらに大規模な並列処理を実現できる。

パイプラインのステージは各アルゴリズムに対応するが、それらが図2-2-1のように互いにデータを介して接続されていると考える。

この流れ図に沿って次から次とデータが流れてくる場合、各ステージごとの処理は矢線の順序関係さえ守ればよく、どんどん新しいデータに対して実行することができる。即ちアルゴリズムごとのパイプライン処理が可能である。

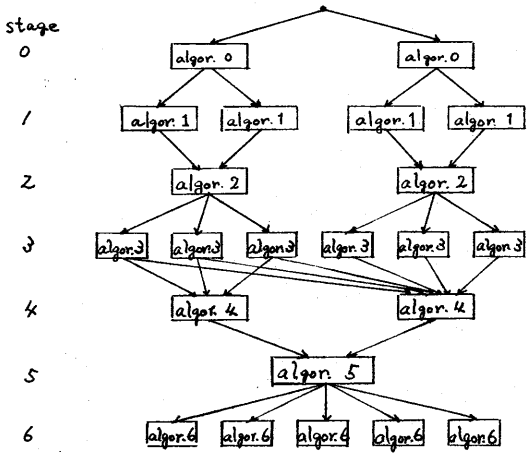

図 2.2.1 パイプラインSAMDシステムの流れ図例

(・図における矢線はデータの流れを表わすと共に処理の順序関係も表わしている。)

問題は多数のプロセッサがこのパイプライン化されたSAMD処理をどううまく制御して行くかということであるが、それについて考察してみる。

プロセッサを何に対応させるべきかを考えると次の3つに分類される。

(1) <プロセッサのデータ従属>

データごとに必要な数のプロセッサを割当てそのデータに関する処理が終了するまで従属させる。

(2) <プロセッサのアルゴリズム従属>

プロセッサごとにアルゴリズムは固定し、データがその位置を変化させながら処理を順次受けて行く。

(3) <プロセッサの独立>

プロセッサは特定のデータやアルゴリズムに従属するのではなく、その時点で適当なアルゴリズムとデータを選択して処理を行なう。

これら3つの場合を比較すると(1)は処理の流れの中に並列度の変化があった場合、その追従が困難である(一般に変化は多い)。(2)は固定割当て方式とも言えるが、アルゴリズムの割当ての予測が難しい場合やデータの流れにばらつきのある場合無駄が出てしまう。従って良い制御方式さえあれば(3)の独立型が最も効率のよい方法と言える。

それを極めて簡単に実現するのが次に述べる自由競争制御方式と呼ぶものである。

§2-3 自由競争制御方式

マルチプロセッサのもう1つの大きな問題は多数のプロセッサをいかに制御しスケジュールを行なうかということである。即ちプロセッサとアルゴリズムとデータの3つをうまく組合せなければならない。

これにはいろいろな考えがあるが、§2-2で述べたように(アルゴリズム, データ)の組を各々プロセッサに指定することでスケジュールすることと考える。

ここで問題となるのは、パイプラインが能率良く流れるのは、パイプの各ステージに要する時間が等しい時なので、うまく制御して、各アルゴリズムの複雑度に比例した数のプロセッサを割当てて、パイプラインの流れを円滑にする必要がある。

これを極めて簡単に実現するのが、ここに述べる自由競争制御方式である。

〔定義〕 自由競争制御方式とは

(1) アイドルになったプロセッサは自分で次の(アルゴリズム, データ)の組を探す。

(2) 仕事が終了するときに、同期カウンタの値に応じて、必要ならば、関連したデータ

をまとめて、適当な（アルゴリズム，データ）の組を作成する．

このように失業したプロセッサが次々と仕事を求めて行くことで、全体として、時間のかかる部分に多数のプロセッサが集中し、パイプラインが能率よく流れて行く．（その様子はシミュレーションで確かめており、§6で詳しく述べる．）

またこの時、完全な自由競争ではなくて、適当な優先度をつけることで細かい制御も可能である．

例えば

- 特殊なI/Oなどの装置につながっているプロセッサは、それを使用するアルゴリズムを優先的に実行する．
- 実時間，オンライン的な仕事は優先する

etc.

§2-4 パイプラインSAMD処理方式の特徴

(1) <並列度を極めて大きく取ることができる>

§2-1でも述べたように、SAMD方式は、SIMDの抱括的な拡張と考えることができるが、それをパイプライン化したものは、SIMDとMISDの両方の性質を持つことになる．これらの並列性は互いに直交性を持っているので積で効き、並列度は極めて大きくなる．

(2) <可変構造パイプラインである>

一般にパイプライン方式はいろいろなレベルで行なわれるが、通常は異なった処理装置の組合せで、パイプラインの構造も固定である．

しかしパイプラインSAMD方式では、パイプラインの構成要素として完全なプロセッサを考えており、それらは対等に扱えるため、可変構造となり、汎用性を持つことができる上に、データの質、量などのばらつきに対して、柔軟に対処できる．

(3) <制御が容易>

自由競争制御方式は、スケジュールが与えられるのではなく、各プロセッサが自分で取りに行くという分散制御の形式なので、並列度がいくら高くても制御ネックにならない．

(4) <信頼性が高い>

一部のプロセッサが故障しても、システム全体としては、少し能力が落ちるだけで、処理が続行できる．

(5) <システム柔軟性>

プロセッサ数，その他の追加，削除などの変更に対して、柔軟性があり、必要に応じたシステム構成が可能．

§2-5 パイプラインSAMD処理方式の適用範囲

<この方式により速度が改善されるもの>

Single Algorithm-stream Multiple Data-stream というSAMDの定義からわかるように、同じアルゴリズムを非常に多数のデータに対してくりかえし用いる場合に適している．

またパイプライン処理は§1-2にも述べたように、時間並列性を空間並列性に変換する代表的手段でもある．

従ってパイプラインSAMD処理方式は、並列度を持つ分岐統合のあるデータの流れ（即ち図2-2-1のようなもの）の上を次々にデータが流れるような構造のプログラムを持つ応用に適している．

具体的な例を思いつくままにいくつか例を挙げると．

- パターン認識，画像処理
- ゲームの次の一手，トリー探索
- 情報検索，広義連想
- 組合せ問題

etc

<従来のものとの親和性（両立性）>

通常のSISD処理をマルチプロセッサでマルチプログラミングすると、それはMIMDに見える．パイプラインSAMD処理でも、各プロセッサごとに1つのSISD処理を割当てて従来通り使うことはもちろん可能である．しかも本当のSAMD処理とSISDとを混在させて実行することも、なんらさしつかえない．

このように、従来のプログラムとの親和性が十分にあることも本方式の適用範囲を広げることができる．

§3. TOPSTAR（パイプラインSAMD用アーキテクチャ）の設計と試作

この章では、§2で提案したパイプラインSAMD処理方式を実現できるハードウェアアーキ

テクチャを設計し，それが現在のＬＳＩ技術の上で極めて少ないチップ数で容易に作成できることを示す．また実際に作成したパイロットモデルについても述べる．

§３－１　パイプラインＳＡＭＤ処理に必要な機能

(1)　＜並列度をアーキテクチャで制限しない＞
これが最優先の目標．

(2)　＜メモリとプロセッサの結合＞
一般にパイプライン処理は，データは全時間を通じてすべてのアルゴリズムで順序処理されるがアルゴリズムとプロセッサが完全に独立だとすれば，各データはすべてのプロセッサからアクセスできなくてはならない．
従ってメモリとプロセッサは完全結合にするか，または部分結合の場合はアルゴリズムとプロセッサの独立性に若干の制限を設ける必要がある．

(3)　＜メモリの全体又は一部に対する排他的アクセス処理＞
これは特にスケジューリングのための同期カウンタに対するアクセスなどのＣＳ（Critical Section）問題を解決する上で，必要不可欠である．

上に述べた機能を中心に，パイプラインＳＡＭＤ処理を簡単に実現できるような方式を設計したので次に述べる．

§３－２　ＴＯＰＳＴＡＲ（TOkyo univ. Pipeline SAMD Transaction ARchitecture）の構成．

パイプラインＳＡＭＤ処理を実現するアーキテクチャとして，§３－１で述べたようなことを考慮しながら次のような方式を設計した．

＜論理的構成＞
概念的にはプロセッサとメモリとの完全結合空間分割型結合であり，メモリアクセス競合の解決はメモリ側が行ない，スイッチのオンオフはプロセッサ側が行なう．制御の部分分散型の結合方式である．

＜具体的構成＞
スイッチの部分は具体的にプロセッサ側及びメモリ側にそれぞれあるＤＭＡＣ（ＤＭＡコントローラ）により同期を取りながら互いに相手をＩ/Ｏと見なして通信を行なう．
具体的な結合図は図３-３-１のようになる．
以下このＴＯＰＳＴＡＲの共有メモリのアクセス制御，特に２つのＤＭＡＣの同期の取り方及びＣＳ問題の解決について述べる．

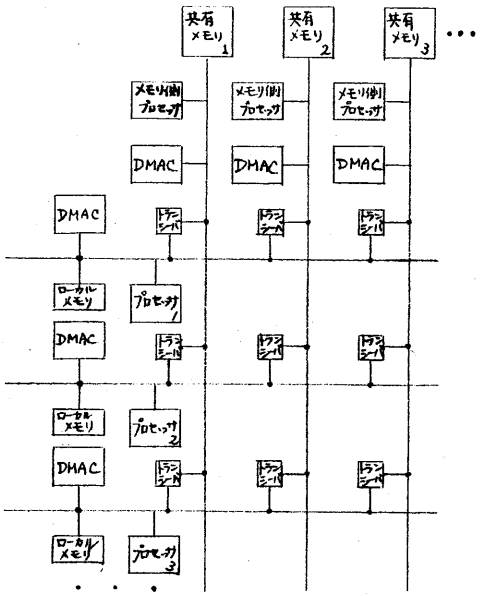

図３-３-１　ＴＯＰＳＴＡＲの具体的構成

§３－３　ＴＯＰＳＴＡＲの共有メモリアクセス制御

ＤＭＡ結合の詳細は図３-３-１のようになっている．ここに用いるＤＭＡＣはインテル社のＬＳＩ８２５７である．

＜共有メモリのアクセス制御＞
共有メモリへのアクセスは２回のＤＭＡにより実現する．

(1)　１次ＤＭＡはプロセッサからメモリに対するアクセス要求で，目的のアドレス，バイト数，read/write の区別などをプロセッサ側からメモリ側の固定領域へ送る．

(2)　２次ＤＭＡは１次ＤＭＡによる情報を用いて，本当のメモリアクセスを行なう．
ＤＭＡは１次，２次ともＣＰＵのＤＭＡコントローラの同期を取って行なう．

プロセッサ及びメモリ側プロセッサの各々の動作を次にまとめてみる．

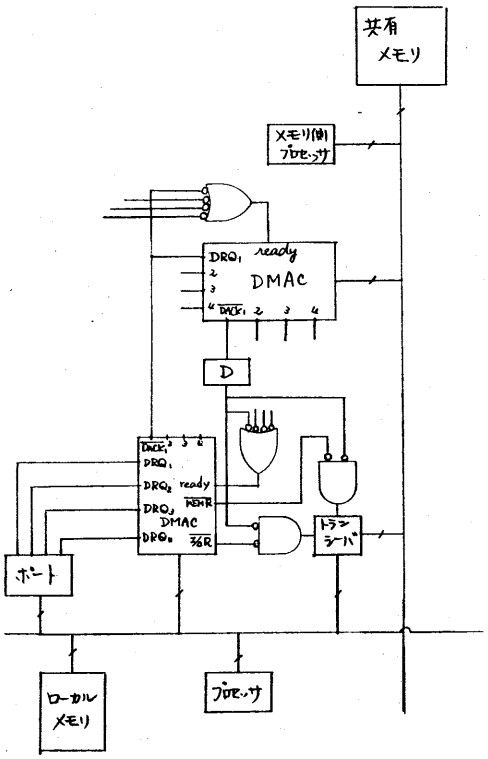

図 3-3-1　TOPSTAR の DMA 結合アーキテクチャ

<メモリ管理プロセッサの動作>

(1) 各メモリバンクごとに連絡用固定領域（DMAバッファ）を用意し、DMAコントローラの内部レジスタに対応する値をセットする.
（1次DMAのための準備）

(2) それらを順に監視している.
（具体的にはDMA終了フラグをチェックする）

(3) どれかに要求があった場合（1次DMAが生じた場合）CSの指定があれば、DMACのモードセットレジスタでそれ以外のプロセッサのフラグを消す.

(4) 固定領域の要求に応じて、DMACの内部レジスタをセットする.
（2次DMAの準備）

(5) DMA終了を監視し、必要に応じて(4)の動作をくりかえす.

(6) (1)に戻る.（CS指定の場合、他のプロセッサのフラグを回復する.）

<各プロセッサの動作>

共有メモリのバンクにアクセス

(1) DMAC内のレジスタに、要求用データの入っているアドレスなどに対応する値をセットする.

(2) DMACのDRQiにI/Oポートを通じて自分にDMA要求を行なう.
（1次DMAの起動）

(3) DMAC内のレジスタに本当のメモリアクセスのための値をセットする.

(4) DMACのDRQiにI/Oポートを通じて自分にDMA要求を行なう.
（2次DMAの起動）

以上の動作をプロセッサ側とメモリ側のプロセッサ及びDMACが各自行なうことで、すべてうまく行く.

ここで重要なことは、2つのDMACの同期の取り方であるが、その詳細を次に述べる.

<2つのDMACの同期>

(1) すべての起動権はプロセッサ側が持つ.

(2) アクセス競合はメモリ側が解決する.

(3) 同期はready信号（本来は遅いメモリの応答用）を利用して行なう.

(4) メモリ側とプロセッサ側を結ぶ信号線は、すべて各々の$\overline{\text{DACK}}$iから出る.

実際の回路図は図3-3-1のようになっているが、2つのDMACの状態遷移と互いの信号との関係を、縦方向に時間の流れを取ってまとめたのが図3-3-3である.

その前に参考のためDMAC Intel 8257の基本的な状態遷移図を図3-3-2に示す.

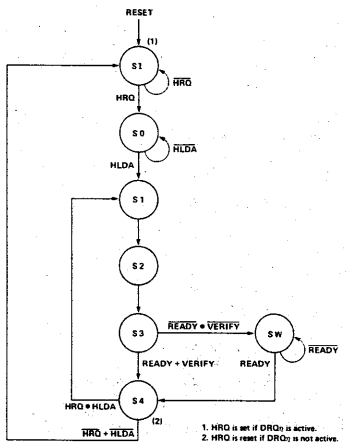

図 3-3-2
Intel 8257の
状態遷移図
（インテルネット
データカタログより
引用）

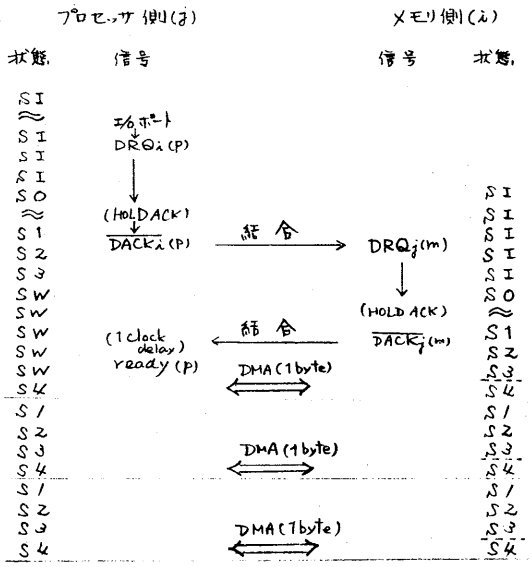

図3-3-3　2つのDMACの同期

(注)　上の方式ではready信号を互いの同期のために用いているが、これは本来遅いメモリ(1クロック以上の応答時間を必要とするもの)のための信号である。従って速いメモリを使用した場合、メモリ側のDMACにもSW状態が入るが(図3-3-3の点線部)，これは、プロセッサ側のready信号をそれに対応する分だけ、遅らせることで容易に解決できる。

また図3-3-3の中の「≈」の記号は、その上の状態が0回以上何回か継続することが、HOLD要求の返事により、ありうることを示す。

信号の矢線関係は相互の因果関係を表わす。

この2つのDMACの同期のタイミングチャートは図3-3-4のようになる。

<プロセッサ側とメモリ側の対称性>

ここで、これまでプロセッサ側、メモリ側と呼んできたのは、処理の流れでの機能上のことであって、そのハードウェア的構造はまったく同じにできることに注意したい。

即ち対称化することによって、1つのモジュールを処理装置でありかつメモリ装置としても使えるということである。

メモリ側のプロセッサは本来メモリ管理を行なうわけだが、この仕事がどれくらい忙しいかによって対称化することの効果が分かれる。

<データ転送速度>

この方式によるデータ転送速度を計算する。

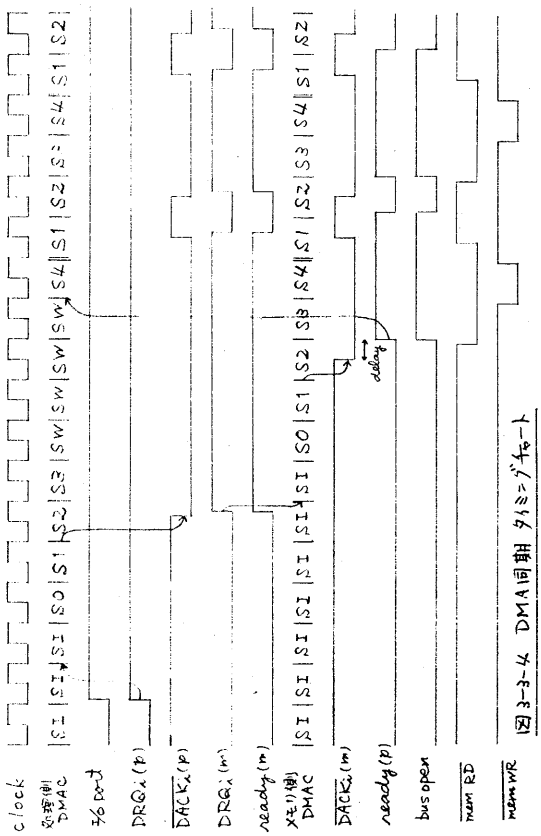

図3-3-4　DMA同期　タイミングチャート

- プログラムモードによるDMAC内レジスタのセット
  1チャネルあたり ---- 9Iクロック
- DMAモードによるデータ転送
  nbytesを送る場合　(8+4n)クロック

合計〔9I+4n〕クロックになる
(1クロック=0.5μsec --- 2MHz)

§3-4　TOPSTARの特徴

(1)　<少数チップ構成>

TOPSTARの大きい特徴は現在のマイクロコンピュータとそのLSIファミリを用いて極めて少数のチップで実現できることである。(具体的には§3-5のパイロットモデル試作を参照)。モジュール間の結合は11本でよい。

(2)　<モジュール性>

インテル8257DMACは1台で4つまでのDMAポートを持っているが、このDMACを必要なだけバスに接続することで、並列度にあわせたシステム構成をモジュール的に実現できる。

(3) <柔軟性>

システムへのプロセッサ，メモリの追加，削除が容易で柔軟性に富む．

(4) <並列性>

モジュールごとのDMACの数は，DMACをメモリ空間に置くことで十分大きくできる．

☆ TOPSTARの問題点

TOPSTARはできるだけ並列度の高い並列処理の実現をめざしているが，規模の大きさに対する問題点もいくつか残っている．

- 完全結合はN²のオーダーで効く．

従って§3-1で述べたように，アルゴリズムとプロセッサの独立性に制限を加えることで，ある程度部分結合にするか，または各バスに複数の処理装置をつけ，時間分割にするなどの工夫が必要になるかも知れない．

- DMACはすべて共通のクロックの下で動く（各処理装置は非同期），従ってクロックの伝達時間の遅れが問題となるような広がりを持つことはできない．

§3-5 パイロットモデルの試作

TOPSTARの基本構成であるDMACによる結合と2台のマイクロコンピュータシステム（8080系）を用いて試作した．

システム本体はテキサスインスツルメント社のTIS-80システムのCPUボードのみを利用，それにDMA用回路を増設した．（ただしこのキットはボード内のDMAが不可能なので，その部分の改造が必要だった）．

<結合のためのLSI，IC等>

- DMAC（8257）
- アドレスラッチ（8212）
- D/FF（7474）
- NOR（7402）

の4素子で各ボードに1組ずつ計8素子用いた．

パイロットモデル全体の回路図は図3-5-3のようになっている．この回路を作成して，メモリアクセス実験を行ない成功した．具体的に言えば，プロセッサ側のTTYからデータを読取りそれを§3-3で述べた同期方法を用いて，メモリ側のRAMに転送を行ない，最後にメモリ側のTTYに出力させたわけであるが，これにより，設計通りDMA結合が働くことを確認した．

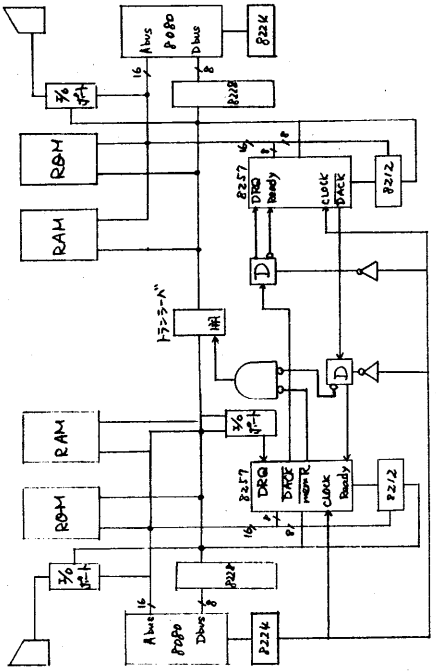

図3-5-1 パイロットモデル回路図

§4 印刷漢字認識システム

この章ではミニコンFACOM-Rと大型計算機HITAC 8800/8700の上で作成した，FSS（Flying Spot Scanner）を入力とする印刷漢字認識システムについて述べ，それによる認識実験の結果この方式が実用的であることを示す．

§4-1 システムの概要

このシステムでは漢字の幾何学的構造に注目して，グラフ構造としてとらえ，認識を行なう方式を取っている．

全体の構成は図4-1-1のようになっているが特徴抽出では，巨視的な情報（大きさ，長さ等）をいかにグラフ構造の作成に反映させるかを，LCSマッチングでは，グラフ構造同志の類似度をいかに判定させるかについて，それぞれ工夫した．

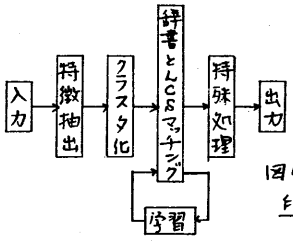

図4-1-1
印刷漢字認識システム

§4-2 特徴抽出

漢字を1次元センサで縦に走査し，その上下の2本の線素の関係を次の2つに着目してコード化する．

(1) 分割数（セグメント数）の変化
(2) 長さの変化（閾値を設ける）

順次作られるコードを字の形のグラフ構造に対応させて，プレックス構造で表現する．

なお上の(1),(2)の変化のない場合，コード化はされず情報量は有効に圧縮される．

(2)の長さの変化に閾値を設けることにより，スムージングは各行ごとに行なうだけで，コード化の安定度を大きく改善できた．また縦方向にも閾値を設けてその範囲内の変化はまとめて1つのコードとすることでもコードの安定化を計っている．

特徴抽出してコード化する様子を「立」という字を例に取って図4-2-1に示す．

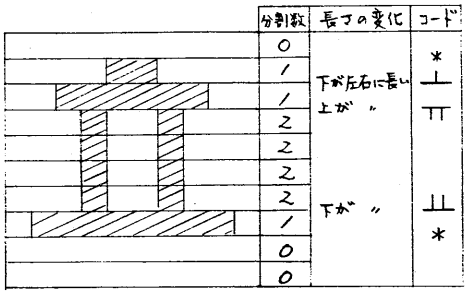

図4-2-1 コード化の例「立」

§4-3 LCSマッチング

本システムでは抽出した漢字の特徴をグラフ構造としてとらえているが，そのグラフ構造どうしの類似度を定量化するものがここに述べるLCS（Largest Common Subsequence）マッチングと呼ぶ方式である．

現在グラフ構造の類似度を求めるのに確立した手法はないが，本方式は計算時間などにおいても充分に実用に耐えうるものである．

マッチングはプレックス構造の1次元化と，その辞書とのLCSマッチングとから成る．

<プレックス構造の1次元化>

プレックス構造の各ノードには，コード，座標，横幅などが含まれるが，図4-3-1のように各ノードに記号をつけてグラフ構造と見ることができる．

これをプリオーダ式に取り出して1次元化すると次のようになる．

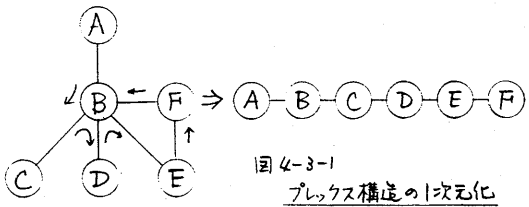

図4-3-1
プレックス構造の1次元化

<LCSマッチング>

〔定義〕 LCS（最大共通部分列）とは，

2つの記号列に対してどちらにも順序をくずさずに含まれる（とびとびでもよい）部分列のうちで最大の長さを持つものをいう．

例えば ABCDEFL と AACFDEGL とのLCSは，ACDEL である．

ここで2つの記号列M，N（長さが各々m，n）のLCSの長さをlとすると

〔定義〕 M，Nの距離は

$$m+n-2l$$

で表わすことができる．

これはediting距離と呼ばれるもので，記号列Mから記号列Nへの変換が最も少ない場合でも（m-l）回の消去と（n-l）回の挿入の合計（m+n-2l）回の操作が必要なことを示している．

類似度としてこのediting距離をそのまま利用してもよいが，これを0～1に正規化した．

〔定義〕 M，Nの類似度は

$$\frac{2l}{m+n}$$ である．

<LCSを求めるアルゴリズム>

LCS及びその長さを求めるアルゴリズムはいくつか発表されているが，ここでは長さだけ求められればよいので，m×nの時間とnの記憶場所から成るアルゴリズムを用いた．

<LCSマッチングの特徴>

(1) $O(n^2)$の計算時間． $\bar{n} = 24.4$
(2) 順序を考慮するので，情報量は大きい．
(3) 任意長コード列の比較が可能
(4) 重みづけがなく情報量を有効に使える．

§4-4 認識実験による評価

認識実験はFSSからの入力データをFACOM-R上のプログラムで特徴抽出してプレックス構造を作成し、それを東大大型センタのHITAC 8800/8700で1次元化し、学習を通して作られた辞書とLCSマッチングを行なった。

類似度が0.7以上のもので最大のものを認識結果として判定するが、この際0.7以上のものがなければ棄却して学習を行なう。

学習とは具体的には辞書の中に追加登録することで、棄却された時以外に誤認識を行なった場合も、データが異常でないことを確かめた上で辞書に加える。

教育漢字881字を入れた辞書から出発し、任意に選んだ100字について、各々5回の認識及び学習の実験を行なった。(入力データは、ゴシック1号活字)

その結果が図4-4-1である。

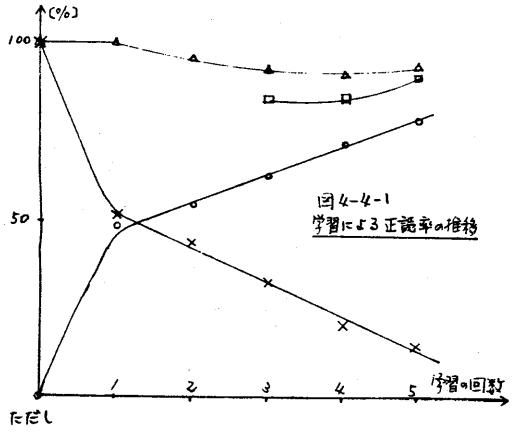

図4-4-1
学習による正読率の推移

ただし

△---棄却後正読率：棄却された文字は除いて正読率を計算

○---完全正読率：棄却は誤りとみなして計算

□---単純正読率：0.7以下の類似度でも棄却せずに最も近いものを判定させた場合

×---棄却率：棄却されたものの率

<プログラムの大きさなど>

FACOM-R上のアセンブリ言語は4.1Kで、これは中間出力、デバッグ用などを含む。大型HITAC 8800/8700とFORTRANは約200行。

認識に必要な時間は特徴抽出が平均25秒(ただしこのうちFSSの読取りに約20秒)。

マッチングにはクラスタ化なしで平均23.2秒ずつかかる。

§5 印刷漢字認識システムのパイプライン SAMD処理

この章では§2で述べたパイプラインSAMD処理方式を上の印刷漢字認識システムに適用することを検討する。

§5-1 印刷漢字認識システムにおける並列性の抽出

§1-3でも述べたように、パターン認識は本質的に多くの並列性を含んでいるので、それをSAMD形式で考えることは一般に容易である。

空間並列性については次のようなものがある。

(1) 字独立性
(2) 行独立性
(3) セグメント独立性
(4) ビット独立性

認識の各段階のアルゴリズムごとに持っている空間並列性をまとめると

- 1行スムージング --- (1)~(4)の独立性
- コード化の前処理 --- (1)~(4) 〃
- セグメント化 --- (1)~(2) 〃
- コード化 --- (1)~(3) 〃
- プレックス化 --- (1)~(2) 〃
- クラスタ化 --- (1) 〃
- LCSマッチング --- (1) 〃

またLCSマッチングはアルゴリズム内並列性も含んでいる。

時間並列性についても各アルゴリズムごとにうまくパイプライン化することができる。

§5-2 印刷漢字認識システムのSAMD化

§5-1で抽出した並列性を活かしてシステムの構成をSAMD化し、その流れを示したのが図5-2-1である。このフローチャートはシステムのデータフローそのものであり、各アルゴリズムの横の添字はデータを示すDE(データアイデンティファイア)で、各々の空間独立性によって分割された時の対応するデータのことである。その各々について(データ、アルゴリズム)の組が作成される。また矢線はデータの流れと共に、処理の順序関係も規定している。

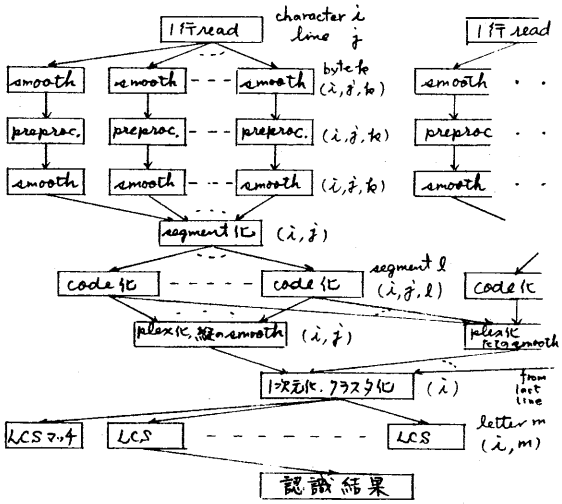

図 5-2-1
印刷漢字認識システム
SAMD data flow chart

<データ構造>

このシステムでのデータ構造は次のようになっている.

- 1行分データ

last line	2重に格納
this line	
OR	
END	
START	

- セグメントごとに

code	
line NO.	座標
bit NO.	
segment NO.	
width	… 横幅

- 同期カウンタ

JOINの時に必要な情報を蓄える

- 個数チェック用 --- 各々1 byte
- プレックス構造作成用(自分より上のすべての行のチェック) --- 行数だけのビット列

<各段階のステップ数>

0~10の10個のアルゴリズムについて述べる.

0---(500kHz)	1---120	2---200	3---120
4---100	5---100	6---140	7---[35]+
[40]+α	8---[20]	9---β	

表 5-2-1

(注) 数字はFACOM-Rのアセンブリ言語のステップ数,[数字]はFORTRANプログラムの行数.
αはクラスタ化 βは特殊処理(共に未実装)
またアルゴリズム0の500kHzは現在試作中の入力システムOCR mind での読取速度である.

印刷漢字認識システムのパイプラインSAMD処理は§3で述べたTOPSTARの上で実現させる予定であるが、その時使用予定のスーパμコンピュータに換算した値などは検討中である.

§6 システムのシミュレーションと評価

この章ではTOPSTARの上でパイプラインSAMD処理を行なう時の管理方式を具体的に述べ,特に§5で述べたSAMD化された印刷漢字認識システムについてその動作をシミュレートして評価する.

§6-1 システム管理方式

TOPSTARを有効に使ってパイプラインSAMD処理を行なうためには,DMA結合の性質をうまく活かす管理方式が望ましい.

それには共通メモリへのアクセス量ではなくてアクセス回数を減らすことが大切である.

パイプラインSAMD処理方式における共通メモリへのアクセスは

- アルゴリズムへのアクセス
- データへの 〃
- スケジュールスタック 〃
- 同期カウンタ 〃

の4つが代表的なものである.

ここでアルゴリズムの入れかえを減少させるため,「なるべく今のアルゴリズムを続けて,次のデータにアクセスする」という戦略を導入する.

これを実現するためには,アルゴリズムごとに別のスケジュールスタックを用意すればよい.

<各プロセッサの動作>

① 今まで行なってきたアルゴリズムに対応するスタックにアクセスする.
(スタックが空ならアルゴリズムも入換)
② 取出したDI(データアイデンティファイア)によりデータにアクセス
③ 処理を実行
④ 処理結果の出力データを格納
⑤ 同期処理

⑥ 必要ならばDエと次のアルゴリズムのスタックに積む（①に戻る）

＜具体的管理法＞

- 1行分のデータはバイト単位で各メモリバッファにインタリーブして入れる．
- 各アルゴリズムの同期カウンタとステージスタックは1つずらして，インタリーブして入れる．
- 入力速度がパイプライン の流速を超えた場合，内部でつまるのを防ぐためにステージの優先度を低くする．etc.

§6-2 シミュレーションプログラム

＜シミュレーションの目的＞

- 自由競争制御方式の下での大量のプロセッサの動きと大量のデータの流れを観察する．
- 具体的に印刷漢字認識システムを適用した例もシミュレートし，メモリアクセス，アルゴリズムの入れ換えなどの頻度を調べる．
- プロセッサの台数を変化させて，レスポンス時間及びスループットの変化を測定し，最適の並列度の大きさを求める．
- 以上のことを通じて，TOPSTARの上で印刷漢字認識システムを作成することの見通しをつけたい．

＜シミュレーション プログラムの構造＞

シミュレーション変数として

- パイプライン SAMD処理に共通なもの
  - (1) プロセッサ台数
  - (2) ステージ数
  - (3) 各ステージごとの平均通過時間
  - (4) 同期方式
- 漢字認識に関するもの
  - (1) 入力速度
  - (2) 1字あたりの行数
  - (3) 1字あたりの画素数

プログラムは東大大型センタのエでPASCALを用いて記述した．全部で240行である．

§6-3 シミュレーション結果と評価

プロセッサ台数に対する，1字認識のために必要な時間を測定し，そのグラフから最適な並列度を求める．図6-3-1に画素数64×64と128×128の例について示す．

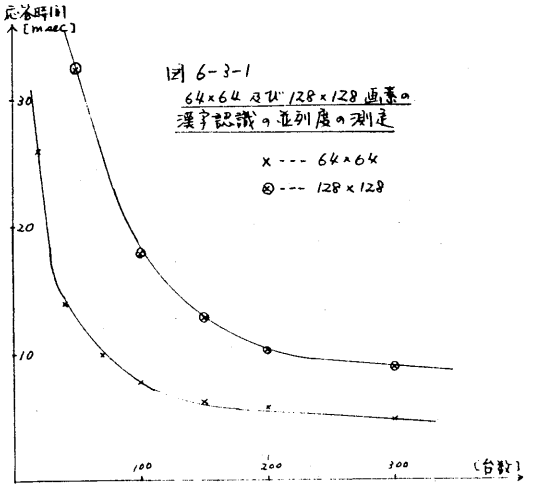

図 6-3-1
64×64 および 128×128 画素の漢字認識の並列度の測定

結論を一言で言うと，以上の研究によって，〃プロセッサを数十台 あるいは百台以上を駆使して漢字を認識することの実現の見通しがある程度ついたということである．

現在Zilog社のZ-80を中心とするシステムを計画中であるが，ハードウェアが大規模となることの影響を今後の課題として解決して行きたい．

また TOPSTAR上でのパイプライン SAMD方式による並列処理の適用範囲を他の人工知能などの問題に拡大し一般化すること及びその並列アルゴリズムの研究なども行ないたい．

＜参考文献＞

1) P.H.Enslow,"Multiprocessors and Parallel Processing,"Comtre corporation

2) R.J.Swan,S.J.Fuller and D.P.Siewiorek,"The Structure and Architecture of Cm*: a Modular Multi-Microprocessor," Computer Science Research Review 1975-1976 Carnegie-Mellon University

3) M.J.Flynn,"Very High-Speed Computer Systems," Proc.IEEE.,vol.54,no.12,Dec., 1966,pp.1901-1909

4) G.A.Anderson,E.D.Jensen,"Computer Interconnection Structures:Taxonomy Characteristics and Examples," Computing Surveys,vol.7,no.4,Dec.,1975,pp.197-213