# プロダクションシステムのための並列マッチング方式とマルチプロセッサによる一評価†

竹内拓二†† 藤田聡†† 相原玲二††
山下雅史†† 阿江忠††

エキスパートシステムを構築する際に用いられる基本的なメカニズムの一つにプロダクションシステムがある．プロダクションシステムは，IF-THEN型のルール記述を行うことにより柔軟なプログラミングを可能にするが，その処理速度は非常に遅い．このことは，記述対象となるシステムの規模の増大や，より高速な応答が要求される分野でのエキスパートシステムの適用に伴って深刻化しており，プロダクションシステムの高速化が望まれている．プロダクションシステムでは，マッチングが全体の処理時間の大部分を占めることが知られており，従来からマッチングの高速化がさまざまな方式によって行われてきている．なかでもマルチプロセッサを用いた手法に関する研究は最近特に注目されてきている．しかしごく一部の研究を除いて，その大半はシングルプロセッサ上でのシミュレーション結果のみによる評価を行うにとどまっている．本論文では，バス結合マルチプロセッサを用いた新しい並列マッチング方式を提案し，既存のマルチプロセッサによる実験，評価を行うことで，提案する方式の有効性を実験的に示す．

## 1. まえがき

プロダクションシステムは，IF-THEN 型のルール記述をプログラミングの基本とするルールベースシステムの一つである．プロダクションシステムには，ルール間の独立性を生かした柔軟なプログラムの記述が可能であること，また人間が持つ断片的な知識を表現するのに適していることなどの長所があり，これらのことからプロダクションシステムは，エキスパートシステム構築のための代表的なツールの一つとして注目されている．

しかし一方では，実行時における処理速度が非常に遅いといった問題点を持ち，このことは，記述対象となるシステムの規模の増大に伴って顕著になる傾向にある．またマッチングがプロダクションシステムの実行時間の大半を占めること，およびこの処理には多くの無駄な動作が存在することが指摘されているなどの点から[1]，プロダクションシステムの高速化にはマッチングの高速化が不可欠であると考えられる．

プロダクションシステムの高速化に関する従来の研究には，大きく分けて二つのアプローチがある．

(1) ソフトウェアによる高速化[1)~4)]．

(2) ハードウェアによる高速化[5)~8)]．

(1)の立場からは，Rete アルゴリズム[2]に代表されるような，無駄なマッチングを減少させるためのアルゴリズムの提案が数多く行われている．

一方(2)の立場は，マッチング処理に内在する並列性を利用し，マルチプロセッサを用いることでマッチングの高速化を図るものであり[5]，この手法は使用するプロセッサ数の点から以下の二つに分類できる．

(a) 多数個プロセッサによるアプローチ…マッチングをデータベースの検索としてとらえ，数千のオーダのプロセッサをサーチマシンとして使用する．代表的なものとして DADO[6] が挙げられる．

(b) 少数個プロセッサによるアプローチ…Rete アルゴリズムを基本とした並列マッチングを，比較的少数のプロセッサによって行う．PSM[7]，MANJI[8] がある．

本研究は，プロセッサ数の点からは(b)のアプローチに着目している．しかし PSM，MANJI がサポートしている並列性とは異なる並列性，すなわちサーチレベルの並列性に着目しており，それに基づいて，バス結合マルチプロセッサを用いた並列マッチング方式の提案を行っている．本稿で提案する方式の，従来の各方式に対する大きな特長は，(1)各処理単位の粒度の細かい“サーチレベル並列性”に注目しているため，各プロセッサの負荷均等化が文献 7),8) よりも容易であること，および (2)通信にバスを用いることで，任意プロセッサからのデータの放送を文献 6) よりも効率的に行うことができること，の二点である．また並列マッチングに関する従来の多くの研究がシン

† A Parallel Matching Algorithm for Production System and Its Evaluation Using a Multiprocessor by TAKUJI TAKEUCHI, SATOSHI FUJITA, REIJI AIBARA, MASAFUMI YAMASHITA and TADASHI AE, (Cluster II (Electrical and Industrial Engineering), Faculty of Engineering, Hiroshima University).

†† 広島大学工学部第二類（電気系）

グルプロセッサ上でのシミュレーションに留まっていたのに対し，本研究ではその効果を，既存のマルチプロセッサを用いて実験的に評価している．

以下2章では，プロダクションシステムの導入を行った後，シングルプロセッサ上での高速マッチングアルゴリズム[1)~4)]の本質的な部分を抽出し，逐次実行型の簡易マッチングアルゴリズム MASS を構成する．MASS の導入は3章以降の議論を簡潔にするためである．次に3章では，アルゴリズム MASS をもとにその並列化に関する検討を行う．ここでは，サーチレベルの並列性に注目した並列マッチングアルゴリズムを新たに提案し，その従来方式との差異を MASS の記述を用いて示す．4章では，この並列アルゴリズムをバス結合マルチプロセッサ上で実行したときの速度向上の効果を，実験的に評価する．最後に5章では今後に残された問題点について言及する．

## 2. プロダクションシステムとマッチング

### 2.1 プロダクションシステム

現在までに最も多く研究対象として取り上げられているプロダクションシステムとして OPS[2)] がある．本研究では基本的に OPS 型のプロダクションシステムを対象とする．プロダクションシステムは，基本構成要素として以下の三つをもつ．

(a) ルールセット：『IF 条件部 THEN 行動部』形式で表されたルールの集合．条件部は一つ以上の条件要素の連接．各条件要素は変数を含むパターン記述である．

(b) データベース：ルールによって参照・更新されるデータの集合．

(c) 推論エンジン．

ルールとデータの記述例 (Ex. 1) を図1に示す．

```
rule: IF    node (X, 1)          CE11
            edge (X, Y)          CE12
            node (Y, 0)          CE13
      THEN  delete  node (Y, 0)
            add     node (Y, 1).

data: node (東京, 1)
      edge (東京, 神田)
      edge (東京, 有楽町)
      node (神田, 0)
      node (有楽町, 0)
```

図1 ルールとデータの記述例 (Ex. 1)
Fig. 1 Ex. 1: An example of rule and data descriptions.

Ex. 1 は，東京駅から JR 線によって行くことのできる駅を求める問題のプログラムの一部であり，駅名 (node) と，隣り合う二つの駅を結ぶ路線 (edge) をデータとして持つ．駅名を表す各データにはその駅が東京駅から行くことができるか否かを示すためのフラグが付加されている．この問題は，『フラグが1の駅Aと，この駅と路線によって隣接するフラグが0の駅Bがあれば，駅Bのフラグを1に更新する』という規則を繰り返しデータベースに適用することにより解くことができると考えられる．図にはこの規則がルールとして記述されている．

推論エンジンの基本動作は次の三つの動作の繰り返しであり，これによって前向き推論を行う．以下ではこの繰り返しをサイクルと呼ぶ．

(1) マッチング：データベースとルールセットを参照し，条件部が成立するすべてのルールとその成立に寄与したすべてのデータの組合せを求める．(得られたルール-データの対をルールインスタンスと呼び，ルールインスタンスの集合を競合集合と呼ぶ．)

(2) 競合解消：競合集合の中から一つのルールインスタンスを選択する．

(3) アクション：選択されたルールインスタンスの行動部に記述された動作を実行し，データベースを更新する．

### 2.2 マッチングとマッチングアルゴリズム

前節で導入したプロダクションシステムにおいてルールの条件部が成立する条件は，

条件(a)：ルールの各条件要素にマッチするデータが存在し（データの不在が成立条件であるような負参照の場合には，データが存在せず），なおかつ

条件(b)：ルールに記述された同一名の変数には同一の値がマッチしている

ことである．図2に図1の例においてマッチングを行

```
競合集合: IF    node (東京, 1)
          edge (東京, 神田)
          node (神田, 0)
      THEN  delete  node (神田, 0)
            add     node (神田, 1).
      IF    node (東京, 1)
            edge (東京, 有楽町)
            node (有楽町, 0)
      THEN  delete  node (有楽町, 0)
            add     node (有楽町, 1).
```

図2 マッチングを行った結果
Fig. 2 Result of matching.

った結果得られた競合集合を示す．

このマッチングはプロダクションシステムの実行時間の大部分（90～99.8%）を占めていると言われており[1]，このことから，マッチングの高速化はプロダクションシステムの高速化に非常に有効であると考えられる．このような観点から，マッチングを高効率化するための数々の手法が従来より提案されてきている[1)~4)]．その本質は以下の二点である．第一のポイントは，データベースを分類し各条件要素に対するクラスタ* を構成しておくことである．ここで各クラスタには，対応する条件要素とパターンがマッチするデータのみが格納される．(図3に図1のデータベースを分類した例を示す．) すなわち各ルールには，そのルールを構成する条件要素数と等しい数のクラスタが対応することになる．このようなデータベースの分類によって，ルールを評価していく際に，条件部の成立に寄与することのないデータとのパターンマッチングを避けることが可能となる．第二のポイントは，各サイクルでの競合集合（マッチングの出力）を保持しておくことである．このことにより，前回のサイクルで更新されたデータに関してのみマッチングを行うだけでデータベース全体に対するマッチングを完了することができ，したがって同一のマッチングの繰り返しを極小化することができる．以上のようなマッチング方式は状態保持法 (state-saving method) と呼ばれ，Rete アルゴリズム[2)~4)] をはじめ，多くの高速マッチングアルゴリズムで用いられている手法である．

逐次型マッチングアルゴリズム MASS (Matching Algorithm using State Saving method) は，Rete アルゴリズムから，3章以降の議論において本質的に必要となる，状態保持の考え方に基づくマッチングの部分を抽出したものである．

$t$ 回目のサイクルにおける MASS の基本動作を以下に示す．

(1) $t-1$ 回目のサイクルで更新された各データとルールセット内の各条件要素との間でパターンマッチングを行い（条件(a)），更新によって影響を受けたルールを検出する．ここで更新とは，データの追加（対<+, data>で表す）またはデータの削除（対<−, data>で表す）のいずれかである．

(2) $t-1$ 回目のサイクルで更新された各データに対して，(1)で検出されたルールに対応するクラスタから，変数に関する条件（条件(b)）を満たすルールとデータの組合せをすべて求め，この結果得られたルールインスタンスの集合（$t-1$ 回目のサイクルでの競合集合の変化分）と $t-1$ 回目のサイクルにおける競合集合から，$t$ 回目のサイクルにおける競合集合を求める．

| | |
|---|---|
| node (東京, 1) | クラスタ ($CEM_{11}$) |
| edge (東京, 神田)<br>edge (東京, 有楽町) | クラスタ ($CEM_{12}$) |
| node (神田, 0))<br>node (有楽町, 0) | クラスタ ($CEM_{13}$) |

**図3** クラスタリングの例
Fig. 3 An example of clustering.

* Rete アルゴリズムでは，$\alpha$ メモリに対応する．

**図4** に MASS の厳密な記述を示す．図4では，上記の基本動作(1)が Step 1～10 で，(2)が Step 11～29 で記述されている．以下では (1)の処理を定数テスト (CT) 処理，(2)の処理を変数テスト (VT) 処理と呼ぶ．このアルゴリズムでは，あるルールのインスタンスは，『そのルールを構成する条件要素に対応するクラスタを一つずつ順に処理対象とし，すでに選択されたデータ集合から定まる変数の値と矛盾しないようなデータをそのクラスタから選択する』という操作を繰り返すことにより作られる．クラスタの処理順序は静的に決定されており，実行時には，静的に作られたテーブル（ただしこのアルゴリズム記述では明示されてない）を参照することで処理が進められる．また CT 処理の出力や VT 処理において発生する中間結果を，ここでは Partially Matched rule Instance* (以下，PMI と略す) と呼ぶ．各 PMI は，(a)処理されるべきクラスタ名を表す識別子，(b)演算（追加あるいは削除）の種類を表すタグ，および(c)中間結果を表すデータ部，の三項組である．PMI のデータ部は，そのルールを構成する条件要素の内のいくつかに対応するクラスタからデータを一つずつ集めた組（ある条件要素に対応するクラスタに属するデータはその条件要素を満足することに注意する）であり，その構成法よりあきらかに，任意の PMI のデータ部に関して条件(b)が満たされている．したがって，任意の二つの条件要素 CE1, CE2 に対し，それぞれに対応するクラスタ内のデータ data1, data2 がともに，ある PMI のデータ部に含まれているとき，CE1, CE2 が共通に変数 $x$ を持っているならば，data1, data2 に含まれる変数 $x$ に対する値は同じである．

* Rete アルゴリズムでは，Rete ネットワーク上のトークンに対応する．

```
procedure MASS
Input:  CS[t-1].
        ΔW[t-1].
        all CEM_ij's.
Output: CS[t].
Method.
  begin
1:  CBUFF := φ;          /* CT procesing */
2:  for each CE_ij do
3:    for each element w_k in ΔW[t-1] do
4:      if  w_k.data and CE_ij.data
            satisfy the condition(I) then
5:        id := (identifier of CE_ij);
6:        w := <id, w_k.tag, w_k.data>;
7:        CBUFF := CBUFF∪{w};
8:      end if
9:    end for
10: end for

11: ΔC[t] := φ;          /* VT processing */
12: while CBUFF ≠ φ do
13:   select one element p_k from CBUFF;
14:   CBUFF := CBUFF−{p_k};
15:   if p_k is a rule instance then
16:     ΔC[t] := ΔC[t]∪{p_k};
17:   end if
/* database search */
18:   for each CEM_ij to be examined† do
19:     sig := (p_k.tag)×(CE_ij.tag);
20:     for each data c_n in CEM_ij do
21:       c := c_n∪(p_k.data);
22:       if c satisfies the condition(II) then
23:         id := (identifier determined from
                   p_k.id and CE_ij.id);
24:         p := <id, sig, c>;
25:         CBUFF := CBUFF∪{p};
26:       end if
27:     end for
28:   end for
29: end while
30: CS[t] := CS[t-1];
31: for each element c_k in ΔC[t] do
32:   if c_k.tag is plus then CS[t] := CS[t]∪{c_k};
33:                      else CS[t] := CS[t]−{c_k};
34:   end if
35: end for
    end.

Comment.
CS[t]：t回目のサイクルにおける競合集合．
ΔW[t]：t回目のサイクルにおけるWMの更新動作の
      集合(演算を表す正負タグと更新データの対)．
CE_ij：ルールiの先頭からj番目の条件要素(参照方
      法を表す正負タグとデータに関する記述の対)．
CEM_ij：CE_ijのデータ部にパターンマッチするデータ
      の集合(クラスタ)．
ΔC：競合集合の変化分．

† 検索すべきCEMの決定は、PIMのもつ識別子とコン
パイル時に得られるテーブルに基づいて行われる．
```

図 4 Procedure MASS
Fig. 4 Procedure MASS.

## 3. マルチプロセッサによる並列マッチング

Rete アルゴリズムの提案によりプロダクションシステムにおけるマッチング時間は大幅に短縮されたが，大規模なエキスパートシステムや制御分野など高速な応答が要求される領域で用いられるシステムでは，依然その処理速度は十分ではない．したがって，マッチングの高速化は依然として重要な研究課題である．

これに対するソフトウェア側からのアプローチとして，Rete アルゴリズムを部分的に改良する提案などが行われてきてはいるが[3),4)]，速度向上には限界があると思われる．その一方で，マッチングには本質的に並列性が内在していることが知られており，これらを利用することにより，マッチングの一層の高速化が可能である．そこで本研究では，ハードウェアからのサポートにより並列マッチングを行うことで高速化を図るアプローチを採る．

### 3.1 マッチングにおける並列性と並列マッチング

Rete アルゴリズム（および MASS）では，データベースは条件要素単位でクラスタに分類され，マッチングはクラスタを単位として行う．

ここで条件要素（クラスタ）単位の並列性について考えてみる．元来ルール間には変数による依存関係はないため，異なるルールに所属する条件要素同士ではルール間の OR 並列性により独立に条件部の判定を行うことができる．また，同一のルールに所属する条件要素間には変数による依存関係が存在するが，パイプライン的な処理（ストリーム並列処理）が可能である．これらを**クラスタ間の並列性**と呼ぶ．PSM[7)]，MANJI[8)]は，クラスタ間の並列性を利用した並列マッチング方式を用いている．

しかし，ここでもう一つの並列性が考えられる．個々のクラスタ内のデータ間には，変数などによる依存関係はないため，各データに対する処理は独立に実行できる．これを**クラスタ内の並列性**と呼ぶ．本章ではこの並列性に着目し，クラスタ内の並列性をサポートする並列マッチング方式を提案する．

### 3.2 マルチプロセッサアーキテクチャ

CAM (Content Addressable Memory) などを用いることによってデータベースの検索時間を大幅に短縮することはできるが，これらの特殊デバイスは現在の技術においては非常に高価になりうる．そこで本研究ではより現実的な立場から，マルチプロセッサによるマッチングの高速化に注目する．さらに以下の理由により，数十程度のプロセッサから構成される MIMD 型のバス結合マルチプロセッサを想定する．

まず第一に，2章で述べたマッチングにおける各並

列度はそれぞれ数十程度であることが指摘されており[7]，このことからプロセッサ数は数十程度で十分である．第二に，多くの場合，マッチングに要する検索時間に対する通信時間の比が比較的小さいことが見込めるため，単一の共有バスを用いたプロセッサ間結合方式で十分な性能が得られると期待される．

### 3.3 並列マッチング方式

3.1節で述べたReteアルゴリズム（およびMASS）の各並列性をバス結合マルチプロセッサを用いてサポートするための二種類の並列マッチングを，2章で導入したマッチングアルゴリズムMASSの実行手続きを用いて述べる．これらの方式はそれぞれ，クラスタ間の並列性をサポートする方式（これを **Stream/OR 方式**と呼ぶ）と，クラスタ内の並列性をサポートする方式（これを **Search 方式**と呼ぶ）である．

マッチングアルゴリズムMASSはCT処理とVT処理から構成されており，それぞれに並列実行が可能である．以下の各節ではVT処理の並列実行についてのみ注目し，CT処理の実行については両方式で同一とする．なおCT処理の並列実行手続きは，図4において，Step 2 を以下の文に置き換えることで表される．

**2′ : for** each $CE_{ij}$ **do in parallel**

またマルチプロセッサ上でのCT処理の実現法は以下のとおりである．まず各プロセッサには，あらかじめCT処理の対象とする条件要素を静的に割り当てておく．すなわち $CE_{ij}$. data は各プロセッサの持つ局所データである．CT処理の実行時には，前回のサイクルにおけるデータベースの変化分を表すデータ集合（図4では $\Delta W$）を全プロセッサへ放送し，各プロセッサでは割り当てられた条件要素に関してのみCT処理を行えばよい．すなわちアルゴリズム中のCT処理の部分に現れる変数はすべて局所変数であり，CT処理開始時においてのみ唯一，入力の，各プロセッサの $\Delta W[t-1]$ 中の各変数への大域的なデータの書き込みが行われる．

#### 3.3.1 Stream/OR 方式[7),8)]

VT処理時には，クラスタ間の並列性により，CBUFF内の個々のPMI同士は，そのそれぞれが処理対象とするクラスタに対する処理を独立に実行できる．つまり，一つのPMIと一つのクラスタを一つの**プロセス**とすることで，これらの並列処理が可能になる．ここで各クラスタ内の処理は逐次実行される．Stream/OR方式の各プロセッサ上での処理手続きは，図4において Step 18 : の前に以下の文を挿入することで表される．ただしCT処理には前節で述べた方式を用いる．

17.1 : **if** no CEM's be examined exist **then**
17.2 : **go to** Step 13
17.3 : **end if**

また各変数はいずれも局所変数であり，Step 25 : のPMIのCBUFFへの追加のみが大域的なデータ書き込み（他のプロセッサの局所変数 CBUFF に対する書き込み）を行う．したがって正確には Step 25 : は以下のように置き換えられる．

25.1 : **for** all CBUFF of other processors
25.2 : CBUFF := CBUFF ∪ {p}
25.3 : **end for**

次にこの方式のマルチプロセッサ上での実現法について述べる．PSM, MANJI とも共有メモリアーキテクチャを採用しているが，その実現法はそれぞれ異なる．その本質的な違いはプロセッサスケジューリングの仕方にある．すなわち，前者が発生したプロセスの動的な割当を行っているのに対して，後者は実行前にクラスタ間の並列実行可能性を調べプロセスの静的な割当を行っている．各実現法のシミュレーションによる比較の結果，静的割当を採用した場合，動的割当に対して速度が数十%程度低下することが示されている[8)]．

#### 3.3.2 Search 方式

Search 方式の手続きは，図4の Step 20 : を以下の文で置き換えることで表される．ここでCT処理は，Stream/OR 方式と同一の方式を採る．また各変数はいずれも局所変数であり，Stream/OR 方式と同様に Step 25 : を拡張する．

20 : **for** each data $c_n$ in $CEM_{ij}$ **do in parallel**

マルチプロセッサ上へは以下のようにして実現する．まず第一に，全プロセッサにすべてのクラスタを静的に割り当てておく．CT処理後，各クラスタの要素（データ）を各プロセッサの局所メモリ（すなわち各 $CEM_{ij}$）へ均等に分配する．また，処理中に発生したPMIはすべてバスを通して全プロセッサへ放送される（Step 25）．各プロセッサはこのPMIに対応するクラスタ内のデータに対するVT処理を行う．これにより，個々のプロセスに対する処理を全プロセッサ上で並列に実行することができる．

## 4. 評　価

### 4.1 理想的な状態におけるマッチング方式の比較

各方式がサポートする並列性は異なるパラメータ（Stream/OR 方式ではクラスタ数，Search 方式では各クラスタ内のデータ数）に依存するため，それぞれの方式に要する処理時間の大小関係は，取り扱う問題によって異なる．しかし直感的には，多数のルールを記述した場合 Stream/OR 方式が，多数のデータを取り扱う場合に Search 方式が，それぞれ高速なマッチングを可能にする傾向にあると思われる．Ex. 1を基本とした簡単な例を用いて示す．ただし本節では各方式における負荷分散の状況を比較することが目的であるため，通信時間などを除いた理想的な状況を仮定している．（通信時間を考慮した評価は4.2節で行われる．）また，マッチングアルゴリズム MASS は CT 処理と VT 処理の2フェーズから構成されていたが，CT 処理は両方式で同一の処理法を採用するため，ここでは VT 処理に要する時間のみについて比較する．

両方式における処理時間を算出する際に仮定した，VT 処理における単位時間と並列実行環境となるマルチプロセッサについて述べる．

単位時間の設定：一つの PMI とクラスタ内の一つのデータとのマッチングに要する時間を1単位時間とする．またこれ以外の処理に要する時間をここでは0単位時間とする．

マルチプロセッサのモデル：バス結合マルチプロセッサシステムを想定する．システムは，先に設定した1単位時間を1周期とする大域クロックを持ち，全プロセッサはこのクロックに同期して VT 処理を行う．ここで通信などに起因する無駄時間は0とする．比較する2方式は，いずれも最適な（処理時間を最小とする）スケジューリングを行った状況で処理時間を算出する．また Stream/OR 方式は，プロセスをプロセッサへ動的に割り当てる方式[7]を想定する．

（仮定終わり）

多数データを想定し Ex. 1のデータ数を 21（node 数5，edge 数 16）に増加した例（Ex. 2），および多数ルールを想定し Ex. 2においてルール数を4に増加させた例（Ex. 3）のそれぞれに，Stream/OR 方式および Search 方式を適用したときの VT 処理の処理速度向上比を**図5**，**図6**に示す．ここでプロセッサが1台の時における VT 処理時間の合計を1としている．

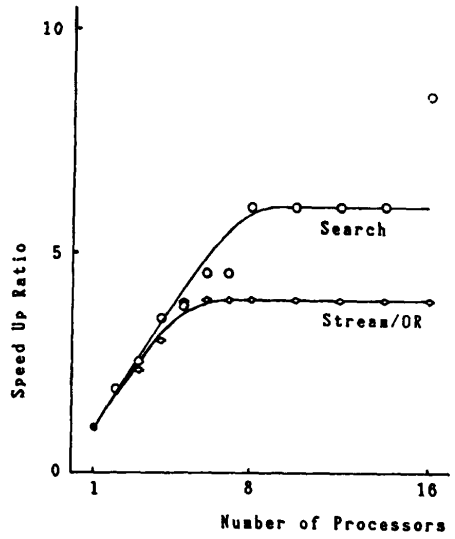


**図5** Ex. 2 における VT 処理の並列実行
Fig. 5 Parallel processing of VT in Ex. 2.

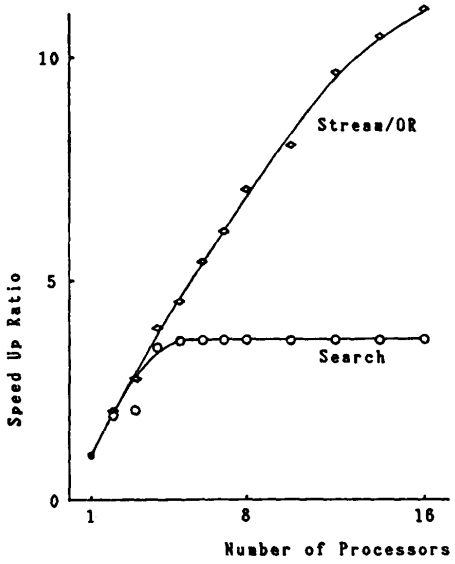


**図6** Ex. 3 における VT 処理の並列実行
Fig. 6 Parallel processing of VT in Ex. 3.

一般性のある定量的な比較を行うのは難しいが，文献7）では，実在するいくつかの大規模な（ルール数 1,000 程度の）エキスパートシステムの分析により，Rete アルゴリズムを用いたときの条件要素間の並列性が，平均で15〜20程度であることが示されている．これは Rete など高速なマッチングアルゴリズムの採用によってデータベースアクセスが十分局所化されることを示すと同時に，Stream/OR 方式を採用した場合，並列処理による高速化の効果が高々十数倍程度し

か期待できないことを意味している．これに対し筆者らの提案する Search 方式では，各クラスタ内のデータ数が十分（具体的には用意されるプロセッサ数以上）多いときには，（通信のオーバヘッドを除けば）ほぼ台数分の高速化の効果が期待できる．したがって，プロセッサ数が十数台程度であれば Stream/OR 方式でも十分効果が期待できるが，それ以上の台数（例えば 32 台）でさらに高速化しようとすると，各クラスタ内のデータ数が十分大きければ，Search 方式の方が有利であると考えられる．

### 4.2 マルチプロセッサ上への実現と評価

3 章で提案した MASS を基本とする Search 方式の並列マッチングアルゴリズムを，マルチプロセッサ上へインプリメントした．実験にはマルチプロセッサシステム UNIP[10] を使用した．UNIP は，マスタ-スレーブ型のマルチプロセッサシステムであり，1 台のマスタプロセッサ（以下，単にマスタと略す）と 32 台のスレーブプロセッサ（以下，単にスレーブと略す）から構成され，全プロセッサは共有バスによって接続されている．各プロセッサには Z-80 A を用い，局所メモリとして 16 Kbyte RAM が装着されている．

まず UNIP 上での Search 方式の実現法について述べる．ただし基本的な動作は 3 章で述べたので，ここでは通信方式について述べる．

通信機能として今回使用したのは，(1)マスタから全スレーブへの放送（broadcast）と(2)各スレーブからの処理結果を回収するためのマスタと 1 台のスレーブ間の個別通信（point-to-point）の 2 通りである．本来ならば，任意のプロセッサが放送機能を持つ必要があるが，UNIP では任意のスレーブから全スレーブに対する放送が行えない．そこで，一度マスタが各スレーブの持つ結果をすべて回収し（ポーリング(polling)＋point-to-point），これらの結果をまとめて全スレーブへ放送することでスレーブ間の通信を仮想的に実現している．

実験は，Ex. 2 を直接 C 言語で記述したプログラムを用いて行った．**図 7** に，実測結果と基礎実験を行った結果得られた通信の内訳を示す．

測定結果では，プロセッサ数の増加に伴う並列処理の効果は現われてはいるものの，UNIP ではプロセッサ間の通信速度が遅いため，実行時間の約半分はマスタ-スレーブ間の通信およびポーリングで占められている．ところが一般に全体の処理と通信の割合は，プロセッサとネットワークの能力の比で決まる．そこで，現在商用機として開発されているバス結合マルチプロセッサ[9] 上で得られる性能を，処理能力の換算により算出した．ここで換算には以下の基準を用いた．

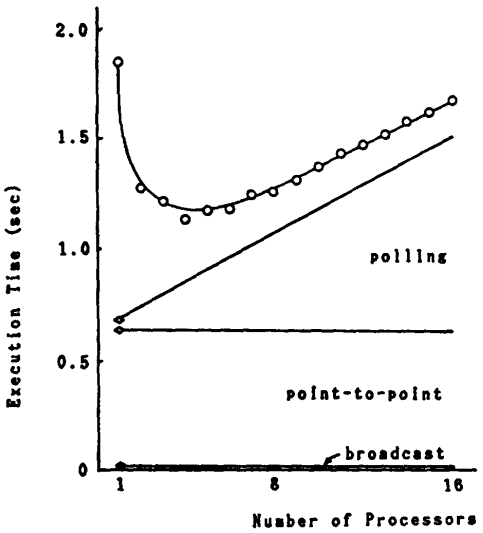


**図 7** Ex. 2 の UNIP 上での実行結果
Fig. 7 Execution results of Ex. 2 on UNIP.

プロセッサの処理能力：

0.4 MIPS (Z-80 A*)→10 MIPS

バスの通信速度：

3.6 Kbyte/sec（実測値）→100 Mbyte/sec

換算結果を**図 8** に示す．（ただし図 7 の通信時間の算出における計算誤差のため，プロセッサ数が 2 台以上での値は多めになっている．）換算の結果から，全体の処理時間における通信時間が占める割合は無視できる程度に小さくなり，バス結合であることがネックにはならないことがわかる．この例の場合，最大 5.5

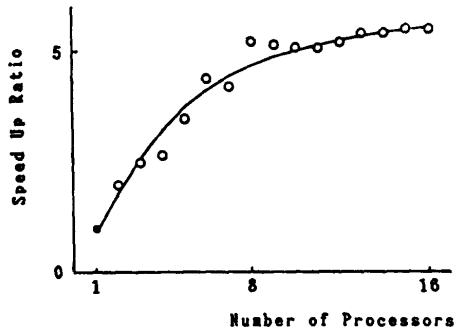


**図 8** 高速なマルチプロセッサ上での性能予測
Fig. 8 Speed up ratio estimation for a high speed multiprocessor.

* Z-80 A のクロックサイクルは 4 MHz である．1 命令を平均 10 クロックで実行すると仮定すると，0.4 MIPS が得られる．

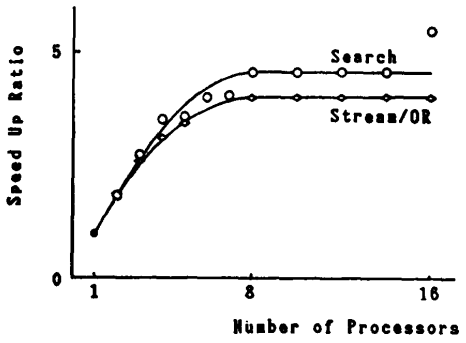


図 9 CT 処理を含めた Ex. 2 の並列実行
Fig. 9 Parallel processing in Ex. 2.

倍程度の処理速度の向上が見込める.

4.1節では，理想的な状況を仮定し VT 処理のみに関する方式の比較を行ったが，次に CT 処理，通信を含めた両方式の比較を行う. Search 方式の実現に用いた C プログラムを使用して 4.1 節で設定した VT 処理での 1 単位時間および CT 処理時間を測定し，これらを 4.1 節で得られた結果（図 5）に適用した結果を図 9 に示す. 差はわずかではあるが，Search 方式がより高速な処理の実現を可能にすることが確認される. ただしここでは，スケジューリングに関する無駄時間は考慮されていない. しかし図 8 に示されるスケジューリングなどの無駄時間を含む実測値とこれらの値がほぼ一致することから，Search 方式ではスケジューリングに関する無駄時間は非常に小さいと考えられる.

PSM ではハードウェアスケジューラの導入を提案しているが，シミュレーション結果によれば並列実行の効果（速度向上比）は，取り扱う問題が元来持つ並列性の 2 分の 1 程度とされている[7]. また MANJI に関しては，データの静的な配置が，問題の持つ並列度を PSM に比べて本質的に低下させることがわかっている[8]. したがって，これらの無駄時間を考慮した場合には，図 9 における Stream/OR 方式と Search 方式の差は一層増大することが予測される.

## 5. むすび

本論文では，プロダクションシステムの高速化を図るアプローチの一つとして，バス結合マルチプロセッサに着目し，データの検索時における並列性（Search 並列性）を利用した並列マッチング方式を提案した. また，提案した方式と類似したアーキテクチャを持つ従来の並列マッチング方式との，基本的性質に関する比較検討および既存のマルチプロセッサを用いた実験結果を報告した. この結果，本論文で提案した方式が，多数のデータを取り扱うシステムでより有効であり，現在商用機として代表的なバス結合マルチプロセッサ上において効率的に機能することが例によって確認された. 本方式は大規模な知識ベースの検索などを伴う応用で特に有効であり，比較的少数のデータをもとに多数のルールによって推論を行うような応用には，従来検討されてきた Stream/OR 方式が適切であると考えられる.

本研究では，並列マッチング方式およびアーキテクチャに関する基礎的な検討を行った. しかし実験に用いたマルチプロセッサではメモリ容量に制約があったため，非常に小規模なプログラムしか取り上げられなかった. このため評価に関してはケーススタディの域を脱しきれておらず，今後，各種の状況を想定した例による比較検討，あるいは実際のエキスパートシステムを用いた検討などを含めた総合的な評価が不可欠である.

**謝辞** 本研究の一部は財団法人マツダ財団助成金および，文部省科学研究補助金奨励研究(A)による.

## 参考文献

1) Mcdermott, J., Newell, A. and Moore, J.: The Efficiency of Certain Production System Implementations, in Waterman, D. A. and Hayers-Roth, F. (eds.), *Pattern Directed Inferrence Systems*, pp. 155-176, Academic Press, New York (1978).
2) Forgy, C. L.: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).
3) 荒谷，百原，田町：プロダクションシステムのための高速パターン照合アルゴリズム，情報処理学会論文誌，Vol. 28, No. 7, pp. 768-775 (1987).
4) 田野，増位，坂口，舩橋：知識ベースシステム構築用ツール EUREKA における高速処理方式，情報処理学会論文誌，Vol. 28, No. 12, pp. 1255-1268 (1987).
5) 石田：プロダクションシステムと並列処理，情報処理，Vol. 26, No. 3, pp. 213-225 (1985).
6) Stolfo, S. J.: Initial Performance of the DADO-2 Prototype, *Computer*, Vol. 20, No. 1, pp. 75-83 (1987).
7) Gupta, A.: *Parallelism in Prodnction Systems*, Pitman Publishing, London (1987).
8) Miyazaki, J., Amano, H., Takeda, K. and Aiso, H.: A Shared Memory Architecture for MANJI Production System Machine, *Proc. 5th IWDM '87*, pp. 354-368 (1987).

9) 天野：マルチプロセッサ型スーパコンピュータ，信学誌，Vol. 70, No. 12, pp. 1263-1274(1987).
10) 相原，阿江：マルチマイクロプロセッサによるソート/サーチエンジンの試作，情報処理学会論文誌，Vol. 26, No. 2, pp. 349-355(1985).

(昭和63年4月14日受付)
(平成元年1月17日採録)

**竹内　拓二**（正会員）

昭和61年広島大学工学部第二類（電気系）卒業．昭和63年同大大学院博士課程前期修了．工学修士．同年(株)立石電機入社．現在，同社システム研究所に勤務．ワークステーションのマンマシンインタフェースに関する研究開発に従事．

**藤田　聡**（正会員）

昭和60年広島大学工学部第二類（電気系）卒業．昭和62年同大大学院博士課程前期修了．現在，同課程後期在学中．3次元集積回路を想定した並列処理アーキテクチャの研究に興味をもつ．電子情報通信学会会員．

**相原　玲二**（正会員）

昭和56年広島大学工学部第二類（電気系）卒業．昭和61年同大大学院博士課程修了．工学博士．現在，広島大学工学部第二類助手．マルチプロセッサシステムの設計，製作ならびに性能評価の研究に従事．電子情報通信学会会員．

**山下　雅史**（正会員）

昭和49年京都大工学部情報卒業．昭和52年同大学大学院修士課程修了．昭和55年名古屋大学大学院博士課程修了．工学博士．豊橋技術科学大学助手を経て，現在，広島大学工学部第二類助教授．この間，昭和61年より約1年間，カナダサイモンフレーザー大学客員助教授．マルチプロセッサシステムの負荷分散問題，画像処理の基礎，組合せ問題の研究に従事．電子情報通信学会会員．

**阿江　　忠**（正会員）

昭和39年東北大学工学部通信卒業．昭和44年同大大学院博士課程修了．工学博士．東北大学助手，広島大学助教授を経て，昭和57年広島大学教授（工学部第二類計算機工学教育科目担当）となり現在に至る．この間，昭和49年より1年間，仏グルノーブル大客員研究員．現在，主として，分散処理システム（マルチプロセッサおよびLAN）の設計，製作ならびに性能評価の研究に従事．電子情報通信学会，ACM，IEEE各会員．