三次元集積回路を想定したプロダクションシステムの並列処理について†

藤田 聡†† 山下雅史†† 阿江 忠††

プロダクションシステム（以下 PS）は，断片的知識を自然に表現できるなどの点で注目を集めている推論メカニズムの一つである．しかしながら PS の処理時間は知識ベースの大規模化に伴って著しく増大する傾向にあり，その解決が強く望まれている．本論文では並列処理による PS の高速化に着目する．特に以下では，従来の研究の多くで注目していたレベルの処理の並列度が高々20程度と比較的少ないことから，これとは異なるレベルの並列性に注目した議論を行う．本論文ではまず最初に，PS 上での複数ルールの並列実行について検討する．具体的には並列実行を可能とする競合解消戦略を一つ固定し，その戦略に従った効率的な競合解消アルゴリズムを提案する．次に，サーチレベルの並列性とルール実行レベルの並列性を特に生かすことを目的とした並列処理アーキテクチャの提案を行う．提案するアーキテクチャでは内容検索可能メモリがデータ検索に，三次元集積回路がプロセッサ通信のためにそれぞれ用いられており，その PS の高速化に関する効果は，具体的な PS プログラムを用いたシミュレーションにより評価される．

1. まえがき

プロダクションシステム（以下 PS）は，断片的知識を自然に表現できるなどの点で注目を集めている推論メカニズムの一つである．しかしながら PS の処理時間は，その実行が検索を基本とすることなどから知識ベースの大規模化に伴って著しく増大する傾向にある．このことは PS の実用上非常に大きな問題点となっている．

この問題を解決すべく，従来よりマッチングアルゴリズムの改良[1)~3)]や並列処理方式の検討[4)~9)]など，PS を高速化するための様々な面からのアプローチが行われてきた．特に並列処理に関しては，商用のマルチプロセッサ上での PS の実現[7)]や PS プログラムの実行に内在する並列性の検討[5),9)]など広範囲に渡って研究が展開されている．しかしながら，従来から研究されている逐次型の PS 実行モデルの並列度は知識ベースのサイズによらず高々20程度であることが，実験的に確認されている[5)]．これは従来の逐次実行モデルの限界の一つであり，より多くの並列性を抽出するためにはモデル自身を拡張することが必要である．一方で PS 実行のより低いレベルには通常あまり考慮されない細かい粒度を持った並列性も内在する[4)]．このレベルの各処理は一般には均一であり，したがってこの並列性に関しては専用ハードウェア（およびデバイス）の使用による効果的なサポートが可能である．

本論文では PS の並列処理に関する提案を行う．以下ではまず最初に，逐次型 PS 実行モデルとその高速実行を行うための状態保持型照合アルゴリズムについて概観する．次に逐次実行モデルの並列実行型への拡張を試みる*．具体的には PS を並列実行するための戦略を提示するとともに，その戦略にしたがって効率的に並列実行を行うための競合解消アルゴリズムを示す．さらに本論文では，新しく導入された実行レベルの並列性を含め，PS のすべての並列性をサポートすることを目的とした並列処理アーキテクチャの提案を行う．このアーキテクチャでは，現在の技術で十分実現可能なデバイスである三次元集積回路[11)~13)]とデータ検索を高速に行うための内容検索可能メモリの使用を想定している．また提案するアーキテクチャ上での並列処理の効果は，シミュレーションにより評価される．

以下2章では従来の研究の概説を行う．3章では並列実行モデルとその効率的な実行のためのアルゴリズムを提案する．また4章では三次元集積回路を想定した並列処理アーキテクチャの提案を行い，5章ではその評価を行う．最後に6章では今後の課題等について言及する．

2. 従来の研究

2.1 逐次実行モデル

PS のモデルとして現在最も一般的なものは次のよ

† On Parallel Processing of Production Systems Using Three-Dimensional Integrated Circuits by SATOSHI FUJITA, 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）

* 本論文では「PS の並列実行」という言葉を，通常の PS の並列処理である「並列照合」とは別の意味で用いる．

うなモデルである*．このモデルは基本構成要素としてルールベース，データベースおよび推論エンジンを持ち，以下の三つのフェーズからなるサイクルの繰り返しによって逐次的に計算（前向き推論）を行う[1]．ここでルールベース中の各ルールは条件部と行動部からなっており，それぞれ“現在のデータベースにおいてそのルールの条件部が満たされるならば，そのルールの行動部の実行によってデータベースの更新を行ってもよい”ことを表している．また各ルールの条件部は複数の条件要素の連接であり，すべての条件要素が満たされたとき（すなわちその条件要素に適合するデータがデータベース中に存在し，それらが条件要素間の束縛変数制約を満足するとき）そのルールの条件部が満たされていると呼ぶ．

[OPS 型モデルにおける計算]（(1)～(3)を繰り返す）

(1) データベースとルールベースを参照することで，条件部の満たされているルールをすべて検出する．（照合フェーズ）

(2) 条件部の満たされているルール集合の中から，適当なものを一つ選択する．選択すべきルールがなければ停止する．（競合解消フェーズ）

(3) 選択されたルールの行動部を実行しデータベースの更新を行う．（実行フェーズ） □

ルールの表現法としてはいくつかの方式が考えられるが，通常は実用性を考慮して変数の使用を許している．この場合照合フェーズで生成されるのは，データベースによって具現化（instantiate）されたルールインスタンス集合である．以下この集合を競合集合（conflict set）と呼び，ルールインスタンスを単にインスタンスなどと呼ぶ．また推論エンジンに用意されている競合解消戦略は，推論実行中固定されていることが多い．その選択基準としては通常，ルールの構造的な複雑さやインスタンスの条件部を構成する各データの生成時刻などが用いられる[10]．

2.2 状態保持型照合アルゴリズム

Forgy によって提案された Rete アルゴリズム[1]は，OPS 型モデルの照合フェーズを効率的に実行するためのデータ駆動型アルゴリズムである．その高速化に関する工夫の本質は，

(a) データベースのクラスタリングによって，データベースアクセスを局所化すること．

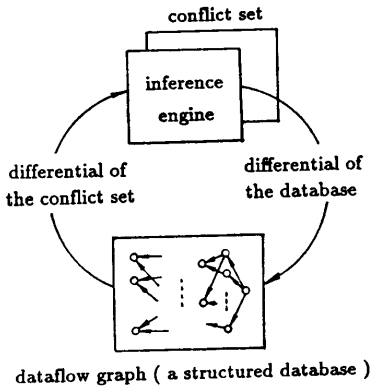


図 1 状態保持型マッチング方式
Fig. 1 State-saving matching method.

(b) 競合集合の保存により，データベースの変化分に対する照合のみで次サイクルでの競合集合が得られること．

(c) 部分的に照合に成功した不完全なインスタンス情報の保存による照合重複の回避．

の三点に集約される（図 1）．なお Rete の後で提案されたいくつかの改良型アルゴリズム[2),3)]においても，以上の各点はほぼ踏襲されている．

3. 並列実行

3.1 並列戦略

本章では OPS 型モデルの並列実行化を試みる．しかしながら一般には，PS プログラムに内在するルール実行レベルの並列性を自動的に抽出することは困難である．これは PS プログラムの理論的な枠組み（特に意味論）を一般的に規定することが困難であることに起因している．一方，PS の宣言型言語としてのメリットを生かすためには，通常の手続き型言語のような並列動作の陽な記述は極力避けるべきである．したがって OPS 型モデルを並列実行化するためには，プログラムの並列性について何らかの具体的（かつ自然）な規定法を与える必要がある．

以下では PS における並列性の一つの規定法を提示するが，規定に先立って，ルールの並列実行に対する一つの要請を明示しておく．

[定義 1]（データベースの矛盾）

任意に固定された PS を想定する．このとき，与えられた初期状態からいかなる逐次実行型戦略によっても到達することのできないデータベースの状態を，矛盾していると定義する． □

* 以下これを OPS 型モデルと呼ぶ．本論文の目的の一つは，OPS 型モデルを並列実行モデルに拡張することである．

```
//Rule-base//

rule1:
if (car status=on sale; price=$1,000)
 & (money amount≥$1,000)
then modify (car status=sold)
  modify (money amount=amount-$1,000)

rule2:
if (video status=on sale; price=$500)
 & (money amount≥$500)
then modify (video status=sold)
  modify (money amount=amount-$500)

//Initial Database//

   (money amount=$1,000)
   (car status=on sale; price=$1,000)
   (video status=on sale; price=$500)
```

図 2 ルールの並列実行が矛盾を引き起こす例
Fig. 2 An example of inconsistency caused by parallel execution of rules.

[要請] (矛盾の回避)

ルールの並列実行によってデータベースが矛盾してはならない. □

データベースを矛盾に導く並列実行例としては，例えば図2のような場合が考えられる．この例の場合，車 ($1,000) を買うというルール1とビデオデッキ ($500) を買うというルール2は初期状態においてはいずれも実行可能であるが，所持金が $1,000 であることからその両方を実行することは許されない．もちろんこの状況は OPS 型モデル上でルールを逐次実行することによって解消可能なものではあるが，しかしながら以下でも述べるように，必ずしも OPS 型モデルに従わなくともこの矛盾を回避することができる．その具体的な一実現法については 3.2 節で述べる．

次に提案する戦略について説明する．この戦略ではルールの実行順序が，静的に定められたルール間の半順序関係（ただし自分自身との順序関係は定義しない）としてユーザにより規定される．ルール実行順序の規定は，ユーザが PS 上の推論を意図的に制御しようとした場合に必要となってくる．図3にルール間の半順序関係の例を示す（以下，このようなグラフをルールの**順序グラフ**と呼ぶ）．図では節点がルールを，有向枝がその順序関係をそれぞれ表している．順序グラフは閉路を含まない有向グラフであり，以下で示す

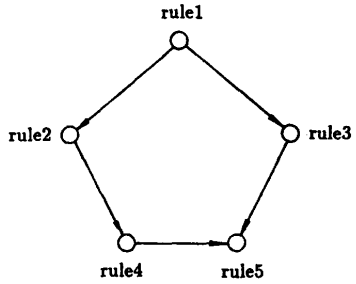


図 3 ルールの順序グラフの例
Fig. 3 An example of rule graph.

戦略中ではこのグラフは，各ルールの実行に関する制御フローを表している．例えば図3の場合，ルール2の実行はルール4や5の実行よりも優先されることになる．（したがって明らかなことではあるが順序グラフ上の閉路は意味を持たない.）また戦略で具体的に考慮されるインスタンス間の順序関係は，ルール間関係から以下のように導かれる．

[定義2] (インスタンス間の順序関係)

インスタンス間の順序関係は，その元になったルール間に順序関係があるときかつそのときに限り存在する．ここで各インスタンスは，その元になったルール名によって識別可能である. □

本論文ではルール実行レベルの並列性の規定を目的として，以下のような並列戦略を定義する．この戦略を効率的に実現するためのアルゴリズムについては次節で議論する．

[並列戦略]

任意の時刻において推論エンジンは，

(1) そのインスタンスよりも大きなインスタンスが存在せず，かつ

(2) そのインスタンスの実行がデータベースを矛盾に導かないような

インスタンスを実行する. □

この戦略は，静的に規定されたインスタンス間の順序関係と先に明示した無矛盾性の要請を満たしている限りは，ルールの並列実行を許すということを表している．なお以下の議論では，グラフの節点に対応するルールのインスタンスのことを単に「節点のインスタンス」などと呼ぶことがある．

3.2 実行アルゴリズム

本節では，3.1 節で提示した並列戦略における各条件を判定するための効率的な手法について述べる．

まず(2)のデータベースの矛盾は，図2の例からわかるとおり，本質的にはあるデータに対する複数イン

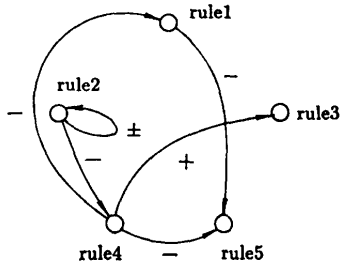


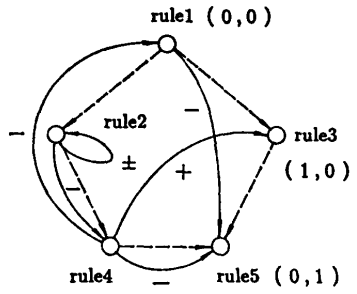
図 4 ルールの依存グラフの例
Fig. 4 An example of rule dependency graph.

スタンスからの異なる要求によって引き起こされるものである．具体的には，インスタンスの実行に伴って別のインスタンスの条件部が満たされなくなることが矛盾の原因である．また一般に PS では，ルールに記述されたデータの型（図2の例では "car"）や定数記述（図2の例では "$1,000"）などによってデータベースを分類することができるため[1]，そのインスタンス同士が矛盾の原因となるか否かがルール単位である程度予測可能であることが知られている[9]．したがってこの依存関係を静的に検出し競合集合中のインスタンスをルールごとに管理しておくことで，インスタンス間の依存関係を実行時に効率的に検出することが可能となる*．図4にデータベースの矛盾に関するルール間依存関係の例を示す[9]（以下，このようなグラフをルールの**依存グラフ**と呼ぶ）．ここで各節点はルールを，有向枝はその依存関係をそれぞれ表している．また有向枝に付記されているラベルは，＋は開始節点のインスタンスの実行に伴って終了節点のインスタンスが新たに生成される可能性のあることを，－は逆に終了節点のインスタンスが消去される可能性のあることをそれぞれ表している（有向枝に＋と－が同時に付記される場合もあり得ることに注意）．なお，図3が制御フローグラフであったのに対して，図4は一種のデータフローグラフである．

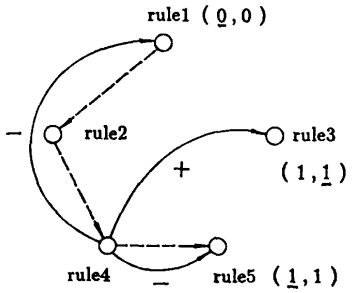
図3，4の各グラフを用いることで，並列戦略の条件判定を以下のようにして効率的に行うことができる．説明のためここでは各グラフを重ね合わせた図5のようなグラフを考える（図では順序グラフの有向枝が破線で描かれている）．また以下のアルゴリズムでは，グラフ上の各節点に，その節点のインスタンスを選択してもよいか否かを示す二通りのラベルR1，R2

* なお文献9）の考察は静的なルール間の依存関係の検出に留まっており，その動的な評価については単にその可能性を示しているだけである．

** その節点自身も先祖節点の一つである．



（a）ルール4のインスタンスの実行（ステップ1）



（b）実行に伴うラベルの変更（ステップ2）

図 5 競合解消アルゴリズムの説明
Fig. 5 Explanation of the conflict resolution algorithm.

(∈{0, 1}) が用意される．図中ではこのラベルは (0, 1)(=(R1, R2)) のように表されている．

[競合解消アルゴリズム]

プロセス 1

（1） 順序グラフを各根節点から走査することで，そのすべての先祖節点** のラベル R1，R2 がいずれも0でありかつインスタンスを持たないような，インスタンスを持つ節点を一つ選択し，その節点のインスタンスを一つ実行する（図5（a））．

（2） （1）で選択された節点を開始節点とする依存グラフのすべての枝に注目する．このとき，

(2-1) 枝のラベルが "－" であり，かつ終了節点が順序グラフ上で自分自身を含む子孫節点の一つである場合には，終了節点のラベル R1 を 1 にする（図5（b））．

(2-2) 枝のラベルが "＋" の場合には，終了節点のラベル R2 を1にする（図5（b））．

プロセス 2

（3） プロセス1の各処理とは独立に，以下の処理を行う．

(3-1) ラベル R1 が 1 である各節点(v)に対し，そのラベルを 1 にする原因となったインスタンスと同時に実行することで矛盾を起こすすべてのインスタンスを消去する．ただし原因となった各インスタンス（唯一ではない）は，消去が完了するまで節点 v 中に保存しておく．消去に関する具体的なアルゴリズムは文献 9) 参照のこと．消去し終えたら節点 v のラベルを 0 にする．

(3-2) ラベル R2 が 1 である各節点に対し，照合の結果その節点に対応するすべてのインスタンスの生成が終了すれば，節点のラベル R2 を 0 にする．

(3-3) ラベル R1，R2 がいずれも 1 である節点に関しては，R2 に関する処理を優先して行う．

□

ステップ(1)のインスタンス実行を一時停止したままでプロセス 1 を繰り返し実行することにより，その時点で並列実行可能なインスタンス集合を具体的に構成することはできる．しかし現実にはインスタンスの実行を一時停止する必要はなく，したがって本アルゴリズムでは，ステップ(1)で生成されたインスタンスから順次（逐次的に）実行していく方法を採っている．

またこのアルゴリズムではプロセス 1 の各ステップが逐次的に実行される限り，前述の並列戦略が正しく実現されることを保証している．何故ならばプロセス 1 のステップ(1)で並列戦略の各条件の判定が行われており，また同ステップ(2)ではインスタンスを実行するごとに各条件に反する可能性のあるすべてのインスタンスの実行を凍結しているためである．ここで凍結の解除は，プロセス 2 の各ステップ(3-1)～(3-3)の実行により，各ルールごとに行うことができる．したがってプロセス 1 の各ステップは逐次的に実行すべきであるが，プロセス 2 の各ステップはプロセス 1 とは独立にしかも各ステップごと並列に実行可能である．

なお次章では，上記の競合解消アルゴリズムを高速に実行し，かつそれ以外の処理の並列性[4),5)] もうまくサポートすることを目的とした並列処理アーキテクチャの提案を行う．本節で述べたアルゴリズムはこのアーキテクチャ上へマッピングされる．提案するアーキテクチャでは三次元共有メモリ[13)] (以下，3-D CM) と内容検索可能メモリ（以下，CAM）の使用を想定している．3-D CM は一種の三次元集積回路であり，構造的には多層化されたメモリである．またこのメモリ内では，層状に構成された各メモリ平面間の大量のデータ転送を，層間結合を用いて高速に行うことができる．したがってこのメモリを用いることにより，高価ではあるがほぼ理想的なマルチポートメモリが実現可能である．

4. アーキテクチャ

本章では提案する並列処理アーキテクチャの概要を示す．処理の概要とハードウェア構成は，それぞれ図 6 と図 7 に示される．ここでは図 6 のようなデータ駆動的な処理を前提としている．

一般にある処理を複数プロセスによって実行する場合には，各プロセスによって共有されるデータをどこに保存するかが問題となるが，図 7 に示すマスタースレーブ型のマルチプロセッサでは，各共有データは直接的に 3-D CM へ格納される．またスレーブプロセッサ群は照合部分と競合解消部分に機能的に分割され

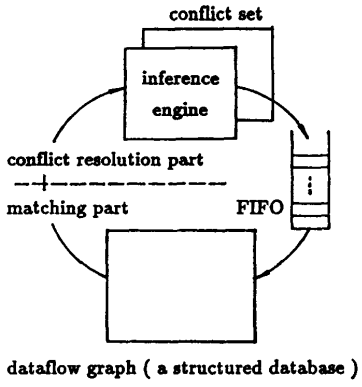


図 6 並列実行の処理フロー
Fig. 6 Outline of the parallel execution.

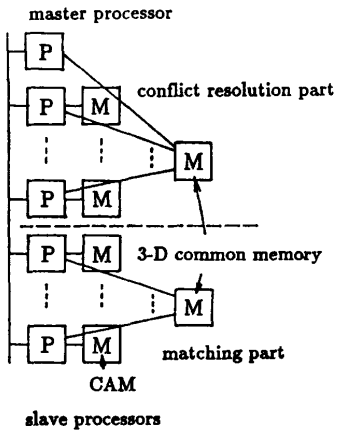


図 7 並列処理アーキテクチャ
Fig. 7 Architecture for the parallel execution.

ている．具体的な構成の方針を以下に示す．

4.1 照合部分

照合部分の説明を行う．まずコンパイル時にデータベースをデータの型やルール中の定数記述などによってクラス化し[1]，各スレーブプロセッサの局所メモリ(CAM)に各クラスタを静的に割り当てておく．照合中のデータベースへのアクセスは基本的にはデータの複数フィールド値をキーとする連想検索であるから，局所メモリを CAM で実現することによって高速なデータ検索が可能となる．一方 PS の照合処理全体は各ルールの条件部を満たすデータ組の検出であり，これはストリーム並列処理によって効率的に実現可能である．ところが実際に稼働率を向上させ，しかもCAMによる高速検索の効果を低下させることなくストリーム並列処理を行うためには，非常に高速かつ低オーバヘッドの通信路が必要となる．

本方式の第一の着眼点は，この通信路を理想的なマルチポートメモリ（すなわち 3-D CM）で実現することである．ただしこの方式は（現在の技術では）構成要素が非常に高価であるという欠点を持つ．なおこれら特殊デバイスの照合高速化に関する効果は，5章で実験的に評価される．

4.2 競合解消部分

次に競合解消部分の構成方針を示す．まずコンパイル時にルール集合を分割し，各スレーブプロセッサに部分ルール集合を静的に割り当てておく[4]．また各プロセッサにはルール間関係の全情報（図5参照）を与えておく．

競合解消は，前節のアルゴリズムのプロセス1をマスタプロセッサで，プロセス2を各スレーブプロセッサでそれぞれ実行することにより実現される．本方式の第二の着眼点は，各プロセッサによって頻繁にアクセスされる競合集合を，3-D CM で直接的に実現することである．ただし競合集合に 3-D CM を用いることの効果は，順序グラフ（図3）や依存グラフ（図4）の性質に強く依存することに注意しなくてはならない．例えば依存グラフの枝数が少ない場合にはプロセス2の各処理の起動回数は減少し，そのためマスタ～スレーブ間の通信頻度も低下することになる．また順序グラフが多くの連結成分からなる場合には，スレーブ上の処理に要求される応答速度はそれほど厳しくないことが予測されるため，ポーリング (polling) などの通信方式でもマスタからの要請に十分対応できると考えられる．したがって実用的には，順序グラフや依存グラフ上の特に高速性の要求されるクリティカルな部分については 3-D CM による通信を採用し，それ以外の部分では別の通信方式を併用する方が，価格性能比の点からは有利であると予測される．

5. 評　　価

照合部分で用いられている特殊デバイス（CAM, 3-D CM）の効果を，PS プログラムに基づいたシミュレーションにより評価した．なおここでは特に，逐次的な競合解消戦略（具体的には MEA 戦略[15]）のもとで正当性が保証される PS プログラムと，そのプログラムの逐次実行とを想定している．これは本実験が照合部分の性能予測を目的としているためである．ただし，このプログラムの正当性を保証するような並列戦略（3章参照）であればそれを適用することもでき，この場合には以下で示される各効果に加えて，さらに次のような効果が期待される：

（1） 競合解消の度に各マッチングプロセス間で同期をとらなくてもよいことによる稼働率の向上．

（2） 並列実行に伴ってクラスタ数やデータ数など単位時間あたりの負荷が増大することによる，負荷不均等の緩和と処理効率の向上．

評価に用いたプログラムは，以下の二つである．

（α） 農夫のジレンマの問題[14]…川の対岸へ渡す狐・が鳥・トウモロコシの数をそれぞれ n（オリジナルは1）に拡張したもの．（ルール数：5　クラスタ数：7）

（β） 猿とバナナの問題[15]…部屋の中に問題の目的と関係ない n 個の対象を散乱させ，照合の時間を意図的に増減できるように初期状態を拡張したもの．（ルール数：20 クラスタ数：22）

またシミュレーションは，以下のような設定のもとで行った．

（1） シミュレーションではマッチングフェーズに要する時間のみを考慮する．またクラスタの配置などに関するスケジューリングは，それぞれの場合でステップ数（後述）に関して最適なものを想定する．

（2） 全プロセッサは，グローバルクロックに同期して1ステップずつ動作するものとする．ただしメモリアクセスに関しては1データをアクセスするのに要するステップ数を1とし，通信に関しては1データを転送するのに要す

るステップ数を1とする.

(3) CAM と RAM の違いは以下のようにして評価する. シミュレーションでは, クラスタあたりのアクセスステップ数を RAM の場合クラスタ中のデータ数とし, CAM の場合照合に成功したデータ数とする. (ただし成功するデータがそのクラスタ中に存在しない場合は, CAM のクラスタあたりのアクセスステップ数を1とする.)

(4) 通信路の違いは以下のようにして評価する. まずバス結合では1ステップあたりシステム全体で高々1データしか転送されないものとする. また 3-D CM では各プロセッサが他のプロセッサに対して, 1ステップあたり高々1データを転送できるものとする.

図8および図9にシミュレーション結果を示す. 図8は CAM の効果を表している. 図では, 問題(α)の解を求めるまでに実行されたサイクル数に対するアクセス回数 (ステップ数) が RAM と CAM のそれぞれについて示されている. ただし同図では, 単一プロセッサによるマッチングの実行と, RAM と同程度の容量を持つ CAM の存在とを仮定している. (なお複数のプロセッサを用いた場合は CAM を分散させることになるが, その際にはできるだけクラスタを分割しないようにする方がアクセス回数を軽減させる上では望ましい. これは本方式ではメモリアクセスをクラスタ単位で行っているためである.) 図から推論中の検索ステップ数が, RAM を使用したときはサイクル数の二乗に, CAM を使用したときはサイクル数にそれぞれ比例していることがわかる. すなわち CAM の使用によってステップ数が大幅に減少している. また一般に高速なデータ検索に関してはハッシュ法の使用も効果的である[7]. しかしながらクラスタ内の全データを検索しなくてはならない場合 (問題(α)はそのような例にあたる) にはその効果は薄く, したがって PS の照合処理では, CAM の使用はハッシュ法よりも強力である.

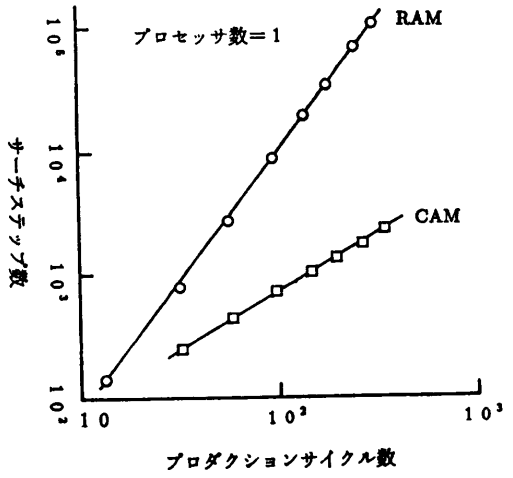


図 8 マッチング部における CAM の効果 (問題 (α))
Fig. 8 Effect of CAM in the matching part (problem (α)).

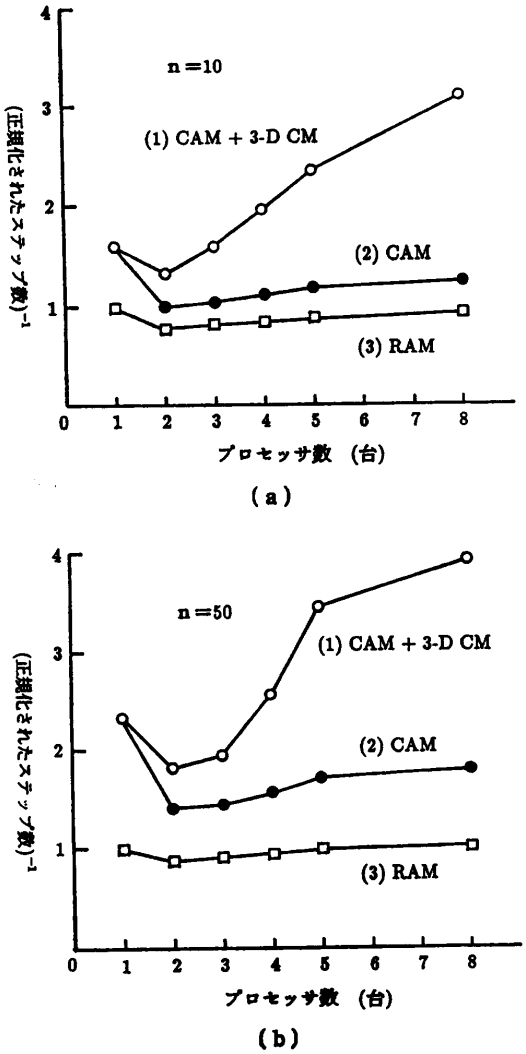


図 9 マッチング部における 3-D CM の効果 (問題 (β))
Fig. 9 Effect of 3-D CM in the matching part (problem (β)).

一方図9は CAM に加えて 3-D CM も効果のある場合を示している. このシミュレーション結果は, 問題(β)の解を求めるために要するステップ数を, (1) RAM を用いたバス型マルチプロセッサ, (2) CAM

を用いたバス型マルチプロセッサ，(3) CAM と 3-D CM を併用したマルチプロセッサ，のそれぞれについて前述の設定のもとで評価したものである．図では横軸にプロセッサ数が，縦軸に RAM を局所メモリとするシングルプロセッサ上での実行に対する速度向上比（総ステップ数の逆数の比）がそれぞれとられている．また同図(a)では問題に関係のない対象数 n を 10，(b)では 50 としている．この図からもわかるように，CAM による高速化の効果は n が大きくなるほど増してはいるが，その効果は 3-D CM の使用によってさらに顕著になっている．これは，3-D CM すなわちマルチポートメモリの使用によって，通信時間および送信待ちのオーバヘッドが大幅に短縮されるためであると考えられる．

6. むすび

本論文ではプロダクションシステムの速度向上を目的として，

(1) 逐次型実行モデルの並列型への拡張，および
(2) 並列性を十分生かすための三次元集積回路を想定したアーキテクチャの提案

を行った．本論文のアプローチは，より広い視野でプロダクションシステムを捕えることによってその一層の高速化や並列化を図ろうとするものである．したがって現在の技術ではその価格性能比は必ずしも良いとはいえないが，しかしながらいずれは，ハードウェア技術の進歩に伴って十分実用化可能であると考えている．

また今後は，ルールの並列実行に関する理論的な枠組みについても検討していく予定である．

謝辞 三次元集積回路のデバイス技術について日頃から有益なコメントを下さいます本学集積化システム研究センター長 広瀬全孝教授，同センター 山西正道教授に深謝致します．また熱心に討論していただいた，相原玲二博士を始めとする計算機工学研究室の皆様に感謝致します．本研究の一部は(財)新機能素子研究開発協会の援助による．

参考文献

1) Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).
2) Schor, M. I. et al.: Advances in Rete Pattern Matching, *Proc. AAAI-86*, pp. 226-232 (1986).
3) 荒屋ほか：プロダクションシステムにおける効率的パタン照合のための連想 Rete ネットワーク表現，情報処理学会論文誌，Vol. 29, No. 8, pp. 741-748 (1988).
4) Stolfo, S. J.: Five Parallel Algorithms for Production System Execution on the DADO Machine, *Proc. AAAI-84*, pp. 300-307 (1984).
5) Gupta, A.: *Parallelism in Production Systems*, Pitman Publishing (1987).
6) Miyazaki, J. et al.: A Shared Memory Architecture for MANJI Production System Machine, *Database Machines and Knowledge Base Machines* (eds. Kitsuregawa, M. and Tanaka, H.), pp. 517-531, Kluwer Academic Publishers (1988).
7) Gupta, A. et al.: Results Parallel Implementation of OPS 5 on the Encore Multiprocessor, CMU-CS-87-146 (1987).
8) Stolfo, S. J. and Miranker, D. P.: The DADO Production System Machine, *JPDC*, Vol. 3, pp. 269-296 (1986).
9) Ishida, T. and Stolfo, S. J.: Toward the Parallel Execution of Rules in Production System Programs, *Proc. ICPP*, pp. 568-575, IEEE (Aug. 1985).
10) McDermott, J. and Forgy, C. L.: Production System Conflict Resolution Strategies, *Pattern-Directed Inference Systems*, pp. 177-199, Academic Press (1978).
11) 黒川，相磯：三次元 IC，情報処理，Vol. 27, No. 7, pp. 718-729 (July 1986).
12) Rosenberg, A. L.: Three-Dimensional VLSI: A Case Study, *J. ACM*, Vol. 30, No. 3, pp. 397-416 (July 1983).
13) Koyanagi, M. et al.: Optically Coupled Three-dimensional Common Memory, *Optoelectronics—Devices and Technologies*, Vol. 3, No. 1, pp. 83-98, Mita Press (June 1988).
14) 五十嵐：OPS 5 vs. COMMON LISP & C, *Computer Today*, Vol. 13, pp. 33-38 (1986).
15) Forgy, C. L. (中間正人訳)：人工知能用言語 OPS 83, パーソナルメディア(株) (1986).

(平成元年 1 月 27 日受付)
(平成元年 4 月 11 日採録)

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，プロダクションシステムの高速化等に興味をもつ．電子情報通信学会会員．

山下　雅史（正会員）

昭和49年京都大学工学部情報卒業．昭和52年同大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，仏グルノーブル大学客員研究員．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．