計算機アーキテクチャ　66－1
（１９８７．７．１０）

リアルタイム・プロダクションシステム ＩＳＡＣのための マルチプロセッサアーキテクチャ

竹内　拓二　　藤田　聡　　相原　玲二　　阿江　忠

広島大学　工学部

プロダクションシステムは柔軟なプログラム記述が可能であるが、大規模なシステムを構築する際、プロダクションの組合せ爆発によりシステムの反応速度が著しく低下することが知られている。　本稿では、推論過程において、状態制御によってプロダクションの組合せ爆発を制御し、さらにマルチプロセッサ上で並列推論を行うことによってシステムのリアルタイム化をめざすＩＳＡＣの概要、状態制御の概念、マルチプロセッサ・アーキテクチャ、製作を開始したＩＳＡＣのプロトタイプ機、および既存のマルチプロセッサを使用した実験による状態制御の効果の確認について述べる。

Multiprocessor Architecture for Real-Time Production System ISAC

Takuji TAKEUCHI, Satoshi FUJITA, Reiji AIBARA, and Tadashi AE

Faculty of Engineering, Hiroshima University
Shitami, Saijo-cho, Higashi-Hiroshima-shi, 724, Japan

We propose ISAC (an Integrated production System Architecture with state-Control)for real-time production systems, which explicitly introduces the concept of state for inference and is realizable on a multiprocessor, to achieve the real-time processing in production systems.

In this paper, we describe the overview of ISAC, the concept of state-control, a multiprocessor architecture for ISAC, a prototype machine and an experimental evaluation to confirm the effect of state-control on a conventional multiprocessor system.

１．まえがき

プロダクションシステム[1][2]（以下ＰＳと略す）は、知識工学の分野、特にエキスパートシステム構築ツールとして注目され、様々な研究が行われており[3][4]、実用的な成果を納めてきている。

ＰＳはプロダクションと呼ばれるＩＦ～ＴＨＥＮ～の集合から記述されるが、このプロダクションによる知識表現は人間の断片的な知識を表現することと親和性が高い、柔軟なプログラムの記述が可能である等の長所を持つ。　これにより、例えば制御用ソフトウェア等、リアルタイム性が必須である分野で使用されるソフトウェアの記述に用いることも注目されつつある[5]。　しかしその反面、実行時におけるプロダクションの組合せ数の爆発的な増加、あるいは競合解消時の制御の飽和問題など、特に大規模なシステムの構築の際には顕著に現れる本質的な問題点を持つことが知られており、実用性のあるシステムの構築のためにもＰＳの高速化が望まれている。

従来、マッチング時(match phase)における処理時間が支配的であるという指摘に対して、ソフトウェアから改良しようとするアプローチがある。例えば、高速マッチングアルゴリズムを提供するＲＥＴＥアルゴリズム[6][7]、あるいはＲＥＴＥの修正版[8][9]等が提案、インプリメントされ、相当の実績をあげているが、今後のエキスパートシステムの規模の増加、リアルタイム処理などを含めた応用領域の拡大を考慮すると、ソフトウェアだけでの処理速度の改善には限界があると考えられる。

一方、高性能コンピュータに対するニーズに応えるアプローチの一つに並列処理技術がある。　特に近年のＶＬＳＩ技術の急速な進歩にともなう高性能プロセッサ、大容量メモリの安価な入手が可能になったことも背景にあり、複数のプロセッサによる処理の分担を行い、高速化をねらうマルチプロセッサシステムが数多く提案、開発されており、その応用範囲は多くの分野におよんでいる。

そこで、ＰＳが本質的に持つ多種の並列性に着目し、マルチプロセッサ上で並列処理を行う手法が提案されている[10]。　その１つに、ＲＥＴＥ等の高速パターンマッチングアルゴリズムと併用しながら並列パターンマッチングを行うアプローチとしてＤＡＤＯ[11]、ＰＳＭ[12]、ＭＡＮＪＩ[8]等がある。　また、マルチプロセッサ上で並列推論を行うことも考えられ、文献[13]では並列推論によるＰＳの高速化とシミュレーション結果等について述べている。　しかし、これらはいずれも競合解消戦略に対する考慮はなく、リアルタイム環境での使用を想定した場合、必ずしも十分な性能を提供しているとは考えにくい。

ＩＳＡＣ（ an Integrated production System Architecture with state Control ）は、リアルタイム処理を可能とすることを目的として提案されたＰＳであり、大きくは以下の３つのアプローチを採用することによってシステムのリアルタイム化を実現する[14][15]。　(1) 推論機構を状態で制御し、プロダクションの組合せ爆発を制御する、(2) 並列推論をマルチプロセッサ上で行う、(3) 高速マッチング処理方式の採用、ならびに３-Ｄ ＶＬＳＩ技術を想定したハードウェアの使用である。　従来の研究は主に、並列推論、高速マッチングの手法を対象とした処理効率の向上であったのに対して、ＩＳＡＣは (1) の手法を取り入れることにより推論空間を縮小し、処理効率の向上をねらうことに最も留意している。　そのため、状態制御アルゴリズムをメタ推論として含みうることをめざし、高機能な競合解消戦略記述能力をユーザに提供する。　状態制御アルゴリズムは、従来提案されてきた競合解消戦略アルゴリズムも記述できる点で、従来のシステムを包含している。

なお、ＩＳＡＣの基本構想[14]、および (3) について[15]は別稿で述べられているので、本稿では (2) を中心に述べる。

以下では、第２章でＩＳＡＣの概要、状態制御、ＩＳＡＣ Language、取り扱うデータについて述べる。　第３章では、ＩＳＡＣのためのマルチプロセッサアーキテクチャの一般的な設計方針を述べる。第４章では、現在製作を行っているプロトタイプ機のアーキテクチャについて述べる。　第５章では、既存のマルチプロセッサを使って状態制御の効果の確認を行ったのでその結果を報告する。

２．ＩＳＡＣ

プロダクション間の非決定性（ nondeterminism ）は、ＰＳが持つ本質的な欠点の一つであり、組合せ爆発を起こす原因になることはよく知られている。これは、ＰＳにおける認識－行動サイクルにおいて競合解消時（ conflict resolution phase ）の制御の飽和問題などを引き起こす原因になり、大規模なＰＳを構築しようとした場合にはシステムの反応速度を著しく低下させる原因にもなうる。 これに対し、ＯＰＳ５[7]ではＬＥＸ，ＭＥＡ、ＯＰＳ８３[16]ではユーザが自由に記述できるように公開する方法がとられるなど、いくつかの競合解消戦略（ conflict resolution strategy ）が提案されてきた[1]。

ＩＳＡＣでは、状態制御という概念を新たに導入し、競合解消戦略の機能の向上を図る。 以下では、ＩＳＡＣの概要と、状態制御、ＩＳＡＣ Language、ＩＳＡＣが取り扱うデータについて述べる。

２．１ 概要

図１にＩＳＡＣの概念図を示す。 ＩＳＡＣでは、ユーザからの質問と、それに関するルールに対して、データ部が管理するデータベースをもとに推論部において推論を行い、結果をユーザに返す。このとき、推論部に状態制御部を陽に付加することによってプロダクションの持つ非決定性の要素を極小化し、システムのリアルタイム化をめざすものである。

また、状態制御のアルゴリズムは知識としてシステムに蓄積される。 これによって、与えられた問題に対して適当な状態制御アルゴリズムを選択できる。 さらに、ユーザが新たにアルゴリズムを記述することも可能であり、これはＩＳＡＣの持つ競合解消戦略の能力を向上する重要なファクタの一つである。 この機構により、状態制御アルゴリズムをメタ知識としてシステムに学習効果を持たせるというねらいもある[17][18]。

リアルタイム環境においては、実時間内のシステムの反応が要求される。 これに対し、ＩＳＡＣは状態制御によるプロダクションの組合せ爆発の制御のほかに、マルチプロセッサ上での並列推論のアプローチを採用し、処理速度の向上を図る。 すなわち、ハードウェアを含めたトータルアーキテクチャを考えることによって、システムのリアルタイム化を実現する。

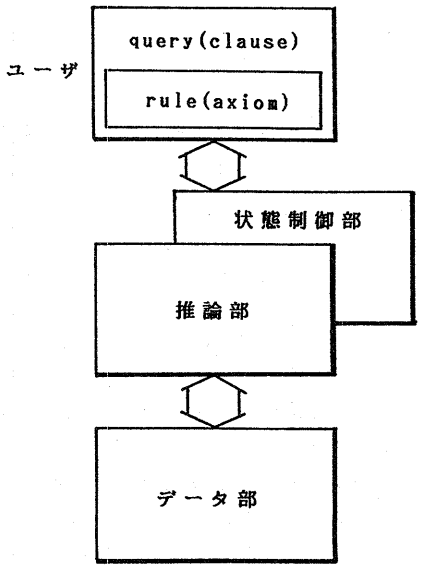


図１．ＩＳＡＣの概念図

２．２ 状態制御

ＩＳＡＣは、推論に対して陽に状態制御を導入する。

＜ Ｘ ＞ ➡ ＜ Ｙ ＞

というプロダクションルールは、もし条件節（以下、ＬＨＳと略す）である Ｘ が満たされると、行動節（以下、ＲＨＳ）である Ｙ が評価、実行されることを意味する。

ここでＩＳＡＣは、ＬＨＳおよびＲＨＳから決定的な要因となる要素をルールから抜出し、陽にルールに付加することによって、プロダクションの持つ非決定的な要素を極小化することを目的とする。従って、先のプロダクションルールのＩＳＡＣでの表現は以下のようになる。

＜ Ｘ，Ｓ ＞ ➡ ＜ Ｙ，Ｓ' ＞

ルールのＬＨＳである Ｘ は状態 Ｓ において適用され、ＲＨＳである Ｙ に書き替えるとともに、状態を Ｓ' に変更するということを意味する。ここで、ルールに対して状態 Ｓ は一つであるが、一般にはベクトルであり、必要な要素はすべて含みうる。

この結果ＩＳＡＣでは

$$\begin{aligned} &< X_1, S_1 > \rightarrow < Y_1, S_1' > \\ &< X_2, S_2 > \rightarrow < Y_2, S_2' > \\ &\qquad\vdots \\ &< X_k, S_k > \rightarrow < Y_k, S_k' > \end{aligned}$$

のようにルールが表現される。 ここで、X_i ($i = 1,\cdots,k$) に関して、ＬＨＳは他ルールあるいは外部事象から非決定的に選択されるが、状態の判定を行うことで発火できるルールを絞りこみ、プロダクション間の非決定的要素を極小化することができる。 これによりプロダクションの組合せ爆発を制御することがＩＳＡＣのねらいである。

２．３ ＩＳＡＣ Language

以上の議論をもとに現在設計しているＩＳＡＣ Language (仮称)でのプロダクションルールの表現は次のようになる。

$$\begin{aligned} &\{ p_1(S) \} X_1 \rightarrow Y_1 \{ f_1(S) \} \\ &\{ p_2(S) \} X_2 \rightarrow Y_2 \{ f_2(S) \} \\ &\qquad\vdots \\ &\{ p_k(S) \} X_k \rightarrow Y_k \{ f_k(S) \} \end{aligned}$$

$p_i(S)$ は状態 $S = S_i$ においてルール $X_i \rightarrow Y_i$ が適用可能かどうかを示す述語である。また、$f_i(S)$ ($i = 1,\cdots,k$) はルールが適用、発火されたことにともなって発生し、状態 S_i を S_i' に書き替える関数である。

ＩＳＡＣ Language では、２．１で述べた状態制御アルゴリズムのユーザによる選択、記述を可能にするため、ルールは問題の公理(axiom)のみが表現され、状態制御アルゴリズムを述語部分で、またルールの適用にともなう状態の書き替え動作を関数部分に持たせる、といった表現法を用いる。

これに対して、従来のプロダクションシステムで用いられている言語(プロダクション言語)では、問題の解法アルゴリズムと公理がルールの中に混在することになり、先の要求が満たされず、システムのリアルタイム化も困難である。

２．４ データ

ＩＳＡＣのデータ部が管理するデータの特徴は、

(1)大量データを許す
　二次記憶装置を想定したデータベースを取り扱う
(2)動的データを含みうる
　リアルタイム環境を想定した場合、データの動的な変化がありうる
(3)データに構造性が有る
　フレーム等の構造データを取り扱う
(4)分割性がよい
　データベースは共通データと複数の分割データに分割できると仮定する

である。

３．マルチプロセッサ・アーキテクチャ

基本アーキテクチャを図２に示す。 状態制御ハードウェア、マッチングハードウェアを導入することでシステムの高速化をねらう。 なお、マッチングハードウェア、および状態制御ハードウェアに対しては検討の余地はあるが、別稿で述べるので本稿ではこれ以上の議論は行わない。

ＩＳＡＣのためのマルチプロセッサアーキテクチャは一般には図３で示される構成で表わされる。ユーザは、ワークステーションを操作し、 質問 (query)を行うとともに状態制御アルゴリズムを選択、あるいは新たに記述する。 システムはこれらの情報をもとにデータ部が管理するデータ上で並列推論を行い、その結果をユーザに返す。

推論部、データ部がそれぞれマルチプロセッサ化されて、ワークステーション、推論部、データ部間はネットワークによって結合される。 推論部では

並列推論による問題解決を行い、データ部では並列サーチ、あるいはデータの分割によって単位プロセッサ当りの管理するデータ量を減少させる、等の手法によりサーチの高速化を目的とする。 状態制御部は推論部から独立し、ユーザの指定に従って推論を制御する。 状態制御は集中型、分散型が考えられるが、現在のところどちらかは特定しない。

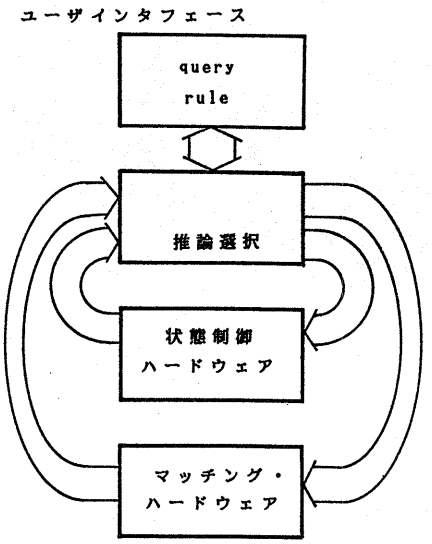


図２．基本アーキテクチャ

ＰＳにおいては、競合解消戦略の違いがＰＳの実行結果に大きく影響することからわかるように、一般に、複数の解が存在し、実行制御法の違いによって得られる解が異なることがある。 従って、並列推論を行う場合、得られる解が特定できず一貫性の保証は困難である。 これは、ＷＭの内容の変更、参照によって発生するプロダクション間の依存関係などが原因となって生じる問題である。 ＰＳの並列推論時の問題に関しては文献[13]で対応法の一つが示されている。

４．プロトタイプ

４．１ 概要

リアルタイム・プロダクションシステムＩＳＡＣの構築の第一段階として、通常のマルチプロセッサにより実現することを考え、プロトタイプ機の構築

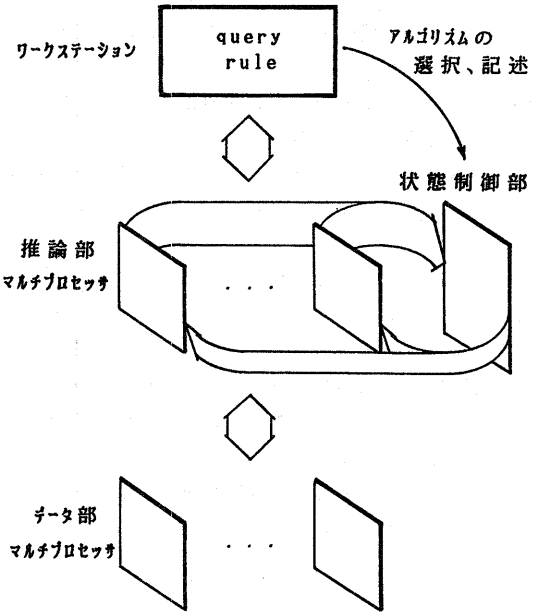


図３．マルチプロセッサアーキテクチャ

を行っている。

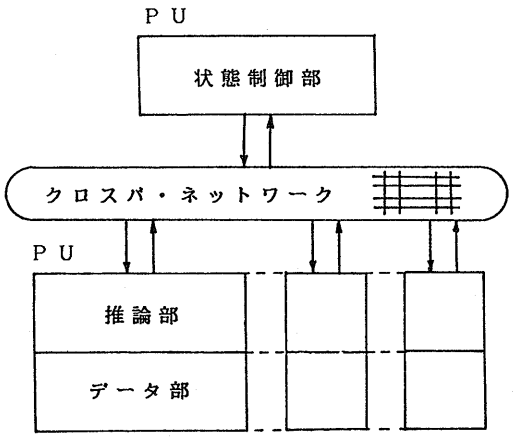
プロトタイプ機では、状態制御ハードウェア、あるいはマッチングハードウェアは存在しない。 従って当面は、状態制御ハードウェアはマスタプロセッサ上でソフトウェアによって集中型の状態制御を行うことによって実現する。 これは、分散型の状態制御に必要な機構が複雑であることが考えられるためである。 また、マッチングハードウェアに関しても同様にソフトウェアで代用する。

プロトタイプ機は、プロセッサとメモリの組から構成される処理ユニットが４台と、処理ユニット間結合網として、４×４クロスバ・ネットワークから構成される。

クロスバ・ネットワークは、ネットワークのコスト（通常、スイッチの必要個数）が$O(n^2)$であり、大規模なネットワークには不利である。 しかし、ＩＳＡＣプロトタイプ製作にあたり、処理ユニット数が４台であること、ネットワーク製作が比較的容易であること等の理由からクロスバ・ネットワークを採用する。

４．２　プロトタイプ機のアーキテクチャ

プロトタイプ機のアーキテクチャは図３をそのままインプリメントすることも考えられるが、(1)並列推論を行う環境での状態制御の効果について確認することを重視する、(2)データの分割性がよい、という仮定のもとにデータ部と推論部を１対１に接続し、図３のデータ部ネットワークは取り除く。従ってプロトタイプ機のマルチプロセッサアーキテクチャは図４のようになる。　即ち、１台の処理ユニットをマスタプロセッサとして割当て、集中型の状態制御と、スレーブプロセッサへの負荷割当て(並列推論の実行制御)を行う。　残りの処理ユニットはスレーブプロセッサとして、分割されたデータ部と推論部を担当し、マスタからの指示に従ってそれぞれが並列推論を行い、状態制御のつどマスタプロセッサへ問合せる。



ＰＵ：Processing Unit

図４．プロトタイプ機のアーキテクチャ

４．３　クロスバ・ネットワーク

製作したクロスバ・ネットワークの外観図を図５に示す。

クロスバ・スイッチの切り換え部分は、マルチプレクサを用いて、スイッチの切り換えは受信側に対して、送信側を選択する。

プロセッサ間のデータ伝送路には　10　Mbps　のEthernet コントローラボードによるシリアル伝送を用いており、高速なシリアル伝送に対して、波形の歪やデューティー比等で十分な特性が得られるように、回路にはＥＣＬを使用している。

プロセッサ間の制御用データ伝送路には当研究室で開発製作したＬＡＮを用いている。　また、通信路の大部分を同軸ケーブルを使用することによって、高速シリアル伝送に耐え得る通信路を確保している。

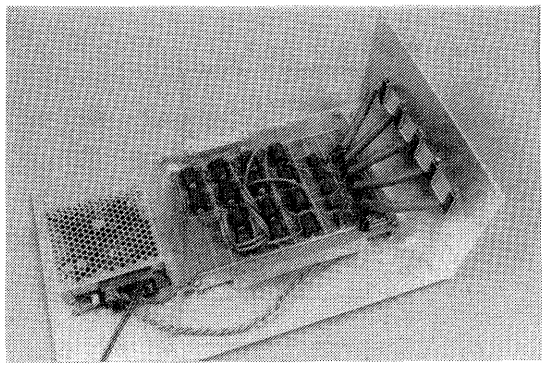


図５．クロスバ・ネットワークの外観図

５．評価

本節では、ＩＳＡＣにおける状態制御の効果について、既存のマルチプロセッサシステムを用いた実験を行って確認する。

ベンチマークテストの一つとして最短径路問題(shortest path problem)を取り上げ、この問題をマルチプロセッサＵＮＩＰ上でインプリメントを行い、処理時間の実測から状態制御の効果を確認する。

５．１　ＵＮＩＰ

マルチプロセッサＵＮＩＰは１台のマスタプロセッサーマスタメモリ（以下ＭＰＵと略す）、３２台のスレーブプロセッサースレーブメモリ（ＳＰＵ）から構成され、各プロセッサは独自のプログラムを実行できる。　ネットワークとしてバス結合網と、パラレルポートを用いたリング結合網を持ち、ＭＰＵ－ＳＰＵ間はスレーブメモリを共有メモリとしてデータの受渡しが行える。　また、システム全体はホストコンピュータに接続され使用される。　図６にＵＮＩＰのマルチプロセッサアーキテクチャを示す。

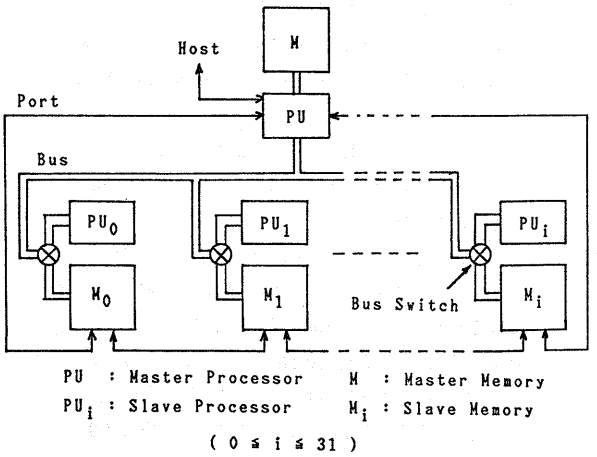


図６．ＵＮＩＰのマルチプロセッサアーキテクチャ

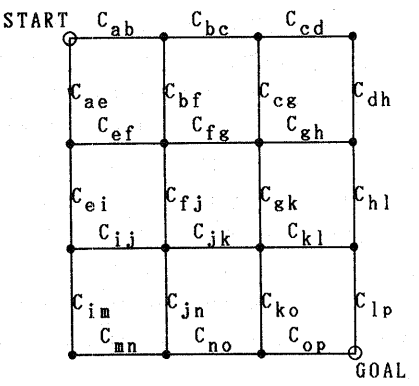


図７．４×４　メッシュグラフ

５．２　ベンチマークテスト

ベンチマークテストとして、図7に示す４×４のメッシュグラフ上での最短径路問題をＵＮＩＰ上で実行した。　最短径路問題は、状態制御の効果が明確に確認できる典型的な例の１つであると考えられることから本稿で取り上げる。　最短径路問題において状態制御アルゴリズムの採用による推論の高速化とは、累計したコストから探索の中止、続行を判断することで探索空間を小さくし、処理を高速化することに対応する。　本稿では、状態制御アルゴリズムの中でもインプリメントが比較的容易な一例であると考えられる解法２つを採用した。　解法Ⅰ：ＤＰ(Dynamic Programming)、解法Ⅱ：ＤＰを使い、なおかつ累計したコストの上限値をあらかじめ設定し、上限値を越える探索は打切る（以下ＤＰ＋BOUND）である。

ＵＮＩＰ上での並列推論の実行モデルについて述べる。　全ＳＰＵは同一のプログラムと与えられる問題のグラフデータを持つ。　ＳＰＵでは、ＭＰＵからサーチの起点となるグラフのノード名と累計したコストをデータとして受け取り、受け取ったノード名から隣接するすべてのノードをサーチし、起点となったノード間の枝のコストを累計コストに加える。　ＭＰＵはＳＰＵからリストアップしたノード名と累計コストを受け取り、累計コストのチェックを行う。　チェックをパスしたデータは待ち状態にあるＳＰＵに渡され、そこで新たなサーチが始まる。　ＳＰＵのスケジューリングは簡単のため、循環しながら順番に割当てている。

問題として与えるグラフは３通り採用し、それぞれ図７のグラフの枝の値 C_{xy} を変えたものである。　表１に各問題をＵＮＩＰ上で４台のＳＰＵを用いて実行したときの処理時間を示す。　また、表２には表１の処理時間から通信時間を除いた値を示す。

PROBLEM 1, PROBLEM 2, PROBLEM 3 はそれぞれ、状態制御の効果が無い、小、大の結果が得られた。

	DP	DP + BOUND
PROBLEM 1	0.246(sec)	0.246(sec)
PROBLEM 2	0.389	0.378
PROBLEM 3	0.390	0.328

表１．ＵＮＩＰ上での処理時間（通信時間を含む）

	DP	DP + BOUND
PROBLEM 1	0.162(sec)	0.162(sec)
PROBLEM 2	0.262	0.216
PROBLEM 3	0.253	0.217

表２．ＵＮＩＰ上での処理時間
（通信時間を含まない）

６．むすび

本稿では、状態を制御することでプロダクション間の組合せ爆発を制御し、さらにマルチプロセッサ上で並列推論を行うことによってリアルタイム化をめざすプロダクションシステムＩＳＡＣのマルチプロセッサアーキテクチャ実現について述べた。　ＩＳＡＣに関しては、文献[14]でＩＳＡＣの基本構想を述べ、文献[15]ではＩＳＡＣのためのハードウェアとして光結合３-Ｄ ＶＬＳＩ技術を想定した高速なパターンマッチングを実現する推論エンジンの基本ハードウェアの提案と、その評価について議論している。

今後、マルチプロセッサ上での状態制御を効率よく実行する計算モデルの骨格、また、そのモデルを有効に機能させるマルチプロセッサアーキテクチャの設計、製作を行っていくが、本格的なシステムの構築に先立ち、より精度の高い評価、シミュレーションが不可欠である。　本稿で行った実験による評価は、状態制御の効果を確認したに過ぎないが、実験、評価を通して、ＩＳＡＣ構築の指標を得たいと考えている。

参考文献

[1] 小林重信; "プロダクションシステム," 情報処理, Vol.26, No.12, pp.1487-1496 (昭60-12).

[2] 辻井潤一; "プロダクションシステムとその応用," 情報処理, Vol.20, No.8, pp.735-743 (昭54-08).

[3] R.G.Smith and R.Davis; "Frameworks for Cooperation in Distributed Problem Solving," IEEE Trans. Syst. Man, Cybern., Vol.SMC-11, No.1, pp.61-70 (Jan. 1981).

[4] 小野典彦、小林重信; "通信/同期機構をもつ分散型プロダクションシステムＰｏ－ＰＳ," 情報処理学会論文誌, Vol.27, No.6, pp.592-600 (昭61-06).

[5] 舩橋誠壽、増位庄一; "制御分野におけるエキスパートシステム," 情報処理, Vol.28, No.2, pp.197-206 (昭62-02).

[6] C.L.Forgy; "RETE:A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, Vol.19, pp.17-37 (1982).

[7] T.F.Lehr; "The Implementation of A Production System Machine," CMU-CS-85-126, (May 1986).

[8] 宮崎他; "プロダクションシステムのための並列アーキテクチャ," 情報処理学会第34回全国大会7P-4, pp.207-208 (昭62-03).

[9] 田野他; "知識処理型ソフトウェアＥＵＲＥＫＡにおける推論機構の高速化," 情報処理学会第31回全国大会5M-7, pp.993-994 (昭60-03).

[10] 石田亨; "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 (昭60-03).

[11] S.J.Stolfo; "Initial Performance of the DADO2 Prototype," IEEE Comp., Vol.20, No.1, pp.75-83 (Jan. 1987).

[12] A.Gupta, et al.; "Parallel Algorithms and Architectures for Rule-Based Systems," 13th Int'l Sympo. on Computer Architecture, pp.28-39 (June 1986).

[13] T.Ishida and S.J.Stolfo; "Towards the Parallel Execution of Rules in Production System Programs," Proc. 1985 Int'l Conf. Parallel Processing, IEEE, pp.568-575 (Aug. 1985).

[14] 阿江忠、相原玲二; "リアルタイムプロダクションシステムＩＳＡＣの基本構想," 信学会コンピュータシステム研究会(昭62)6月発表予定.

[15] 藤田聡、相原玲二、阿江忠; "リアルタイムプロダクションシステムＩＳＡＣのためのハードウェア、信学会コンピュータシステム研究会(昭62)6月発表予定.

[16] C.L.Forgy; " OPS83 User's Manual and Report," tech. report, Production Systems Technology (1985).

[17] 伊藤貴康、松山隆司; "推論ソフトウェアの構成Ⅱ," 信学会誌, Vol.20, No.5, pp.508-516 (昭62-05).

[18] B.Silver; "Meta-Level Inference," North-Holland (1986).