計算機アーキテクチャ　89－2
（１９９１．　７．　１９）

機能メモリ指向の推論マシンのアーキテクチャ

坂井雄介　　橋本勉　　小原清弘　　斎藤延男
東京農工大学工学部　数理情報工学科

大規模なPrologプログラムをインタプリティブかつ高速実行する逐次形推論マシンのアーキテクチャを提案する。このマシンでは、可逆メモリとパターン連想メモリの二つの機能メモリを用いて処理系の高速化を図る。REMは記憶内容の回復機能を持つため、その上に変数の格納領域を取ることにより、バックトラックを高速化・自動化する。PAMはゴール節を与えられると、全ての引数がパターンマッチするような候補節を、内部で引数間並列で高速に検索、出力する。このいわゆるClause Indexingの効果により単一化失敗の回数が減少する。さらに、処理系を三つのモジュールに機能分散し、処理系内の並列性を引き出した。このときの実行の手順についても述べる。

An Architecture of Inference Machine based on ASIC Memories

Yuusuke Sakai　　Tsutomu Hashimoto　　Kiyohiro Obara　　Nobuo Saitou

Faculty of Technology, Tokyo University of Agriculture and Technology
2-24-16 Naka-chou, Koganei-shi, Tokyo 184, Japan

In this report, we propose the architecture of inference machine to execute large Prolog programs interpretively with high peformance. We aim to speed up Prolog processor utilizing two Application Specific Integrated Circuit (ASIC) memories. Those memories are REversible Memory (REM) and Pattern Associative Memory (PAM). REM is assigned to region of variable terms. While backtracking, REM unbinds instances rapidly and automatically, using function to retrieve contents. PAM decreases the number of unification failure, due to search such clauses as applying clause-indexing to each argument. Furthermore, extracting parallelism of processor, we divide it in three function units which execute synchronously.

1．はじめに

Prologは、その論理型言語としての特徴を活かして、おもに知識処理や人工知能の分野におけるプログラミング言語として使用されている。そして、実行速度の高速化を目的としてProlog処理系には多くの改良が加えられてきた。特に、WarrenによるPLMやWAMなどの抽象命令セットを用いたコンパイル技術[1]は、処理系の実行速度を大幅に向上させている。またその後も、コンパイラでの最適化、専用ハードウェア化そして並列処理といった種々の技法が提案されている。

このような現在のPrologの高速化手法は、コンパイルしたプログラムを前提に考えられている。確かに、コンパイラを使うことによってリスト操作やクィーン問題などのトイプログラムを高速に実行することが可能である。しかしながら、知識処理のような大規模なプログラムでは、コンパイル方式は不向きである。この理由としては、プログラムを少し書き換えただけで、再びそれを実行するのに膨大なコンパイル時間がかかることや実行中に規則、つまり自分自身を書換えながら実行を進めてゆくプログラムでは、節の追加・削除が困難であることがあげられる。

このような問題に対して、我々は知識処理などの大規模なプログラムを高速かつインタプリティブに実行する逐次形Prolog処理系のアーキテクチャを提案する。

このアーキテクチャでは、推論実行時間の大半を占める単一化時の候補節検索とバックトラックにそれぞれ専用の機能メモリを用いることにより、個々の処理の高速化を図った。

候補節検索部分は、パターン連想メモリ(PAM: Pattern Associative Memory)と呼ばれる機能メモリを用いて高速化を図る。PAMは、ゴール節を与えられ、単一化可能な節を提示する。この単一化可能性のチェックは、各引数に対して並列に行われる。すなわちPAMは、いわゆるコンパイラの Clause Indexingを、全引数について行うのと同様な処理を、インタプリタに対して提供する。このような機能は、節の数が少ないようなプログラムには効果が少ないが、我々が目的とする実用的な大規模なプログラムの節の検索には効果を発揮する。

バックトラックについては、可逆メモリ (REM: Reversible Memory)を用いて高速化を図る。REMは書換えによって失われた記憶内容を回復することが可能なメモリである。REM上にPrologの変数を格納しておくことで、バックトラック時の変数の未束縛化処理を、1コマンドで高速かつ自動的に実行することが可能となる。

このような機能メモリを用いた個々の高速化に加え、Prolog処理系全体の高速化および並列性を抽出するために、処理系を GIM(General Integration Module)、RUM(Reversible Unification Module)の2モジュールとパターン連想メモリで構成した。GIM は処理系全体の制御を行い、RUMはPAMよりインデキシングされた節の単一化を行う。また、可逆メモリは RUMのメモリ空間の一部に配置して使用する。

以降の構成としては、2章で二つの機能メモリについて述べ、3章では推論マシンの構成とその効果的な実行方式を説明する。さらに4章で推論マシンの実現形態について述べる。最後に、5章でまとめと今後の研究について述べる。

2．機能メモリ

この章では、REMとPAMという二つの機能メモリについて、そのしくみとそれを用いて行う高速化の方法について述べる。

2.1．REM

可逆メモリ(REM)は、書換えによって失われた記憶内容を回復することが可能な機能メモリである。REMは、その記憶内容をエポックと呼ばれる論理的な時間変数と対応付けて管理し、外部からエポックの値を操作することで、以前の記憶内容の回復処理を行う。

REM では、アドレスを与えてデータを入出力する通常の読み書きの他に、次に述べる5つのコマンドを受け付ける。それらは、REM の内部を初期化するイニシャライズ、エポックの値を操作し記憶内容の保存/回復を行うインクリメントエポック(INC EP)とデクリメントエポック(DEC EP)、指定されたエポックから現在のエポックまでを一つのエポックにまとめるマージエポック、および過去のエポックの内容を読み出すパストリードである。コマンドの一覧を表1に示す。また、REMの動作の様子を図1に示す。

REM の記憶内容の回復機能を別の視点からみると、REM はエポック値に対応した記憶内容を持っているようにも見ることができる。ここでエポックの変化をREMの世代の変化と考える。Prolog プログラムの実行経過を変数の束縛/未束縛の状態に注目して考えると、REM上に Prologの変数を格納したとき、記憶内容、つまり変数の値は、単一化処理を区切りとして変化する。

表1　REMに関する操作

コマンド名	実際の動作
インクリメントエポック	エポック値のインクリメント
デクリメントエポック	エポック値のデクリメントし、記憶内容を回復する
マージエポック	現在から指定したエポックまでを1つにまとめ、エポック値を指定した値にする
イニシャライズ	REMの内部を初期化する
パストリード	指定したエポックでの記憶内容を読み出す

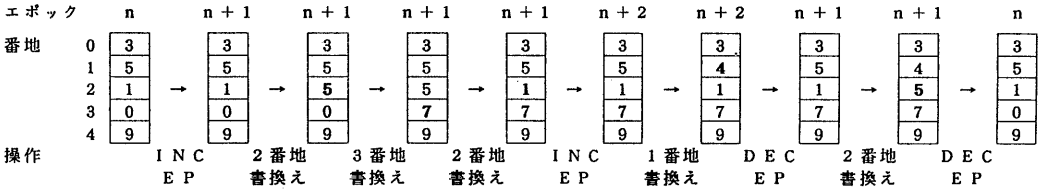


図1　REMの動作の様子

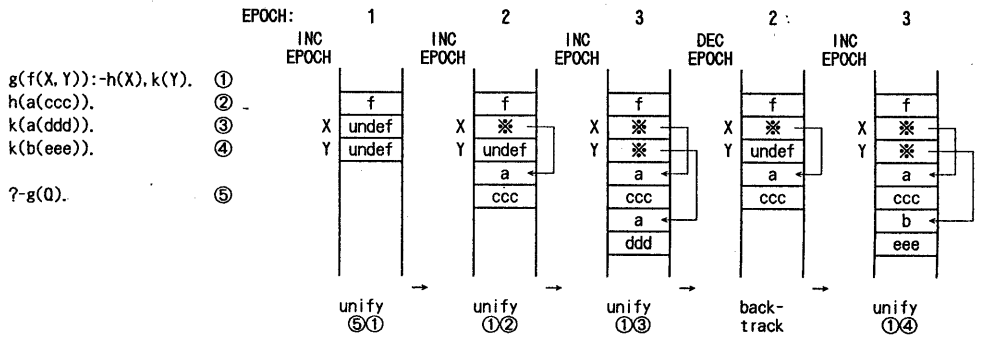


図2　Prologプログラム実行時のREM上のインスタンスの遷移の例

このことから、ある単一化処理と次の単一化処理の間を、REM の世代と対応させることによりエポック操作による変数の世代管理を実現する。すなわち、新たな単一化処理を開始するときには、インクリメントエポックを実行し、REM を次の世代にする。この世代では、その単一化に付随する変数の束縛処理が施される。バックトラック時には、デクリメントエポックで REMを一つ前の世代に戻す。このとき、REM の記憶内容の回復機能により、先ほどの世代での束縛処理は無効となり、変数は未束縛化される。この実行例を図2に示す。

マージエポックは、Prologのカットの処理に対応している。処理系がカットの処理を行った場合、REM に対し、そのカットが出現した節の頭部の単一化が行われた世代までのマージエポックコマンドが与えられる。すなわち、その節の頭部の単一化からカットの直前のゴールの実行までが一つのエポックにまとめられる。このことにより、プログラムの実行がカットを超えて戻るときにも、デクリメントエポックを一回実行するだけで済み、カットを含むプログラムに対しても前述のエポック操作の方式がそのまま適用できる。

REM を用いることの利点としては、処理の高速化・簡素化がある。従来のトレイルスタックに関する処理、すなわち代入変数位置の記録やトレイル作業が専用ハードウェアで高速かつ自動的に行われる。REM を使用してもバックトラック時には記憶内容の復活の処理は一定の時間がかかってしまうが、REM にエポックのデクリメントを指令した後には、プロセッサは REMと独立に処理を続行できることから、プロセッサ側のバックトラックに続く処理と REM内での記憶の回復が同時に実行される。このため多くの場合、変数内容の未代入化処理時間を見かけ上無いものとして考えることができる。

以前に行ったREMの評価[2]では、TTL-ICで実現した REM をProlog処理系に組み込んだ処理系において、バックトラック時間がクイックソートで約2.7%、機械翻訳で約9.4% に短縮することが分かっている。REMを使用したときのバックトラック時間のほとんどが、REM へコマンドを与える時間である。

次に、REM の仕様について考える。考慮すべきパラメタは、プロセッサからアクセス可能なアドレス空間の大きさ（間口サイズ）、回復可能な記憶内容の大きさ（奥行きサイズ）、そして単一化の回数に対応するエポックの最大値、が重要なものとしてあげられる。

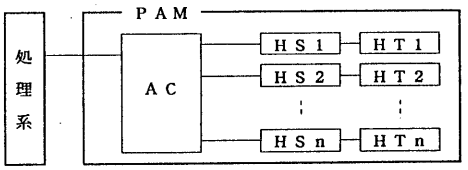
ここで Prolog処理系に対する、REMの各パラメタの値を考えてみる。変数の格納領域の大きさが、間口サイズを超えそうなときには、複数個使用してスタック全体を割り当てることができるので、REM 一個分の間口サイズの大きさは任意でよい。Prologの変数は、単一化をシーケンシャルに行うときには一度に一つの値しか束縛されないので、奥行きサイズは間口サイズの倍あれば充分である。単一化の回数は、扱うプログラムの性質によって異なる。今のところ、周囲に実用的なプログラムが無いため、エポックの最大値を確定できないが、大きめに見積って、これを 4ギガ程度としている。

以上、REM について述べたが、より詳しい説明が文献[3]で述べられている。

2.2. PAM

パターン連想メモリ (PAM)は、ゴール節を与えられると、それと単一化可能な節へのポインタ（節番号と呼ぶ）を出力する機能メモリである。PAM は内部に、節の頭部と節番号の組を保持している。そして、与えられたゴールに対して、単一化可能な節を引数間並列に検索を行いその節番号を出力する。これにより、全引数に対して Clause Indexingをしたのと同等な節が検索される。

PAMの構成を図３に示し、PAMを用いた候補節検索の概念を図４に示す。PAMは連想コントローラ(AC: Association Controller)と複数個のハッシュシーケンサ(HS: Hash Sequencer)から構成されている。HSは節の述語と各引数に対応しており、HS1が述語用、HS2が１番目の引数用というように順次割り当てられている。HSは、与えられた引数とマッチングできる要素をハッシュテーブルから検索する。またACはHSが返してきた情報をまとめて、処理系(CPU)に結果をわたす。



AC : Association Controler
HS : Hash Sequencer、　HT : Hash Table

図３　PAMの構成

PAMには、WRITE、READ、DELETEの各コマンドが用意されている。次に各コマンドの動作を述べる。

WRITE では、まず最初に処理系は節の頭部と属性、そして節に一対一対応する節番号を計算して PAMに与える。属性とは、本来のPrologには無い概念であるが、これは節に付属する情報で、ゴールと同じ属性を持つ節だけを検索するために用いる値である。節の本体は処理系側が保持する。PAM 内では、ACが、入力された節の頭部を各項目（すなわち述語名と引数）ごとに分解し、それぞれ項目長（引数の数＋１）、属性と節番号を付加したタプルを生成する。すなわち、合計で引数の数＋１個のタプルが生成され、対応するHSに転送される。それぞれのHSはタプル中の項目、項目長と属性をキーにしてオープンハッシュを行い、ハッシュテーブルにACから渡されたタプルを格納する。だが、オープンハッシュではテーブル中の特定の場所に格納が集中し、検索時のコストがかかる欠点がある。そこで項目や属性が同じものを語尾変化群という単位にまとめて格納する方法を採用した。

語尾変化群の格納例を図５に示す。H/M 領域とは、語尾変化群の先頭(Head)か構成要素(Member)かを識別するフラグ領域である。COUNT 領域は、自分の属する語尾変化群の先頭や語尾変化群の次の番地にジャンプするためのオフセットを格納しておく領域である。この二つの領域の情報を用いて語尾変化群の頭出しを高速に行うことができる。語尾変化群内では項目は節番号の順にソートされて格納される。

READでは、処理系は PAMに対して質問ゴールと属性を与え、ACはこれを WRITE時と同様な処理を施し、各HSに転送する。HSは項目と属性が一致する語尾変化群

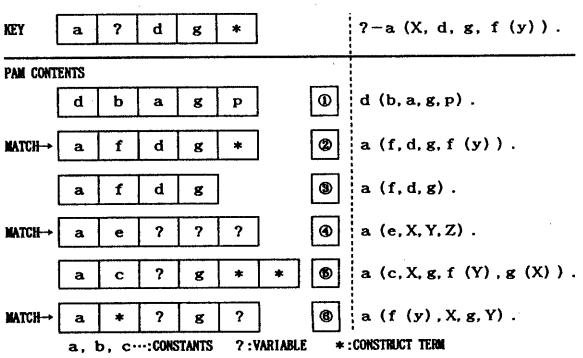


図４　PAMを用いた候補節検索の概念

H／M 領域	COUNT 領域	項目長・属性または節番号
H	6	項目長・属性
引数（述語）項目		
M	－2	5
M	－3	17
M	－4	38
M	－5	64

図5　語尾変化群（ハッシュテーブルの一部）

を探して、そのMemberの先頭の節番号、つまり語尾変化群の中の最小節番号をACに返す。ACは返されてきた節番号を一致照合する。一致照合の様子を図6に示す。一致照合では、ACはまず節番号を比較する。節番号が一致しないときは、その中で一番大きい節番号以上のものをHSに検索させる。これをすべての節番号が同じ値をとるまで行い、一致すると、処理系はその節番号に対応する節を単一化候補節としてゴールとのマッチングを行う。

また、READは節番号とともにそれがハッシュテーブルのどの位置にあったかという中断情報を処理系に返す。PAMには本来の READの他に、この中断情報を使ったREADが用意されている。このコマンドを実行すると、各HSでは見つかった節番号の次のMemberから一致照合を続行する。

この READは2種類あり、1つは、PAMが今出力したばかりの中断情報を使ったREADで、これはシャロウバックトラックが起きた時に次に実行する候補節検索を行うためのものである。もう1つは、処理系から以前の中断情報を受け取ってのREADである。こちらはディープバックトラックが起きた時に先ほど中断した単一化の続きを行うために使われる。

DELETEコマンドでは、処理系は PAMに節番号を与える。PAMはこの節番号をハッシュテーブル内の語尾変化群から削除する。

さて、READコマンドでは、PAM は定数と構造体の第1レベルのマッチングをのみを行い、共有変数の一貫性チェックや第2レベル以上の構造体に対する一致照合は行っていない。第2レベル以上のマッチングと共有変数の一貫性チェックは処理系側が行う。

本来は、PAM ですべての単一化処理の中のマッチングや変数の束縛をやってしまうのが理想的であが、以下の問題が生じてしまう。

変数を PAMで扱う場合の問題としては、変数の一貫性を管理するために何らかのスタック機構が不可欠だということがある。スタック管理機構を持つ機能メモリも考えることはできるかもしれないが、この問題は節の検索とはまったく別のものである。また、構造体の引数のマッチングを PAMでやるためには、HSの処理にレベル情報を加えなくてはならず、PAM での処理が複雑化してしまう。

これらを PAMに実行させると、機能メモリの範囲を超えた処理をすることになる。そこで PAMでは変数の一貫性の管理や構造体の引数は扱わない。

節の格納場所などが現在の仕様とやや異なっているが、以前のPAMの評価[2]では、述語名だけのハッシングで検索をする処理系に対し、PAM を使用した場合には、単一化回数はクイックソートで 73%、機械翻訳で85%、また一番効果があった意味ネットワークで 4%にそれぞれ減少した。

さてPrologに適合するようなPAMの仕様を考えるときに、HSの個数を考慮する必要がある。現在は、Prologのゴールの引数の個数が平均3個であることなどから、HS数は5としている。引数の個数がこれより多い場合には、最後のHSがいくつかの項目をまとめて扱うことにしている。

PAMは現在PGAを使った実現を進めている。現在の仕様と比べて多少の変更はあるが、PAM のより詳しい記述は文献[4]に示されている。

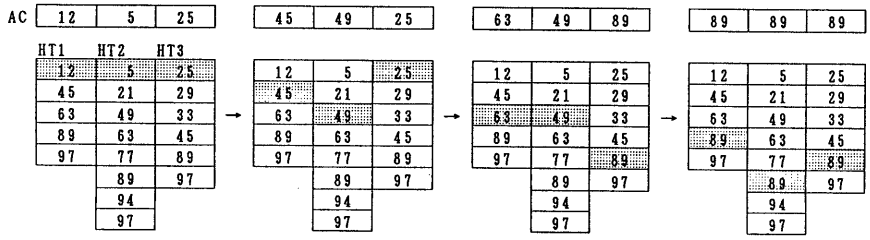


図6　一致照合（この場合は4回の一致照合で節番号89で一致する）

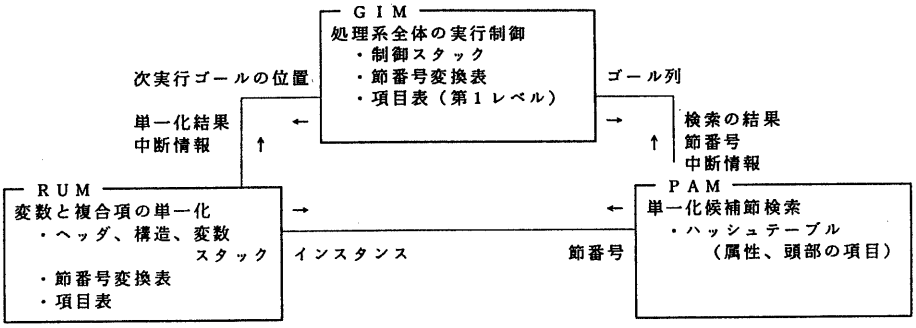


図７　推論マシンの各モジュール

３．推論マシン

この章では、前述した機能メモリに基づく推論マシンの構成、実行方式について述べる。

３．１．推論マシンの構成

推論マシンの構成を図７に示す。推論マシンは GIM、RUMという２つのモジュールとPAMから構成される。ここではGIMとRUMについて述べる。

３．１．１．ＧＩＭ

GIM (General Integration Module)は、推論マシン全体の実行を制御するモジュールである。GIM は、Prolog処理系のスタックのうち、制御スタックの管理をすることで処理系全体の制御を行う。制御のほかに、GIM は外部との入出力、プログラムの内部表現への変換や個々の節に固有な情報である節番号の生成も行う。また GIMは、節の第１レベルの項目を内部形式で格納した表を管理する。これは、GIM が実行制御を行うのに必要な情報である。

GIM の制御スタックには単一化が失敗したときに制御を戻すべき情報としてRUMとPAMの中断情報を格納する。中断情報とは、単一化が失敗してバックトラックが起こったときの戻り番地、いわゆる選択点の情報である。RUM での中断情報は、単一化処理を始めたときのスタックの位置である。一方、PAM の中断情報は、PAMが単一化候補節の節番号を返すときに、PAMのハッシュテーブルの語尾変化群内で現在の節番号が格納されていた次の番地である。つまり、同じゴールに対する次の単一化候補節の節番号の格納番地が中断情報として格納される。２．２節で述べたように、PAMにはこの番地を入力として受け取り、以前の検索を再開するREADコマンドが用意してある。

３．１．２．ＲＵＭ

RUM(Reversible Unification Module)は、単一化処理のうち、変数の束縛と構造体の引数のマッチングを行う。それ以外の単一化処理、すなわち候補節の検索と定数項のマッチングは PAMでの節検索処理で既に終了している。

RUM は、節のインスタンスを格納するスタックとして、ヘッダスタック、変数スタック、構造スタックを管理する。また、単一化に必要な、節の内部表現や節番号から節の位置への変換表など節に関するすべての情報を管理する。

RUM 内のスタックの使用例を図８に示す。ヘッダスタックは、WAM のスタックに相当し、ここには単一化の実行の順に各節の変数のインスタンスへのポインタと RUM自身の制御情報が格納される。構造スタックと変数スタックは、あわせて WAMのヒープスタックに相当する。構造スタックには、構造体の定数部が格納される。変数スタックには変数内容そのものが格納され、バックトラック時の変数の未代入化処理を自動的に行うためにREM上に割り当てられている。

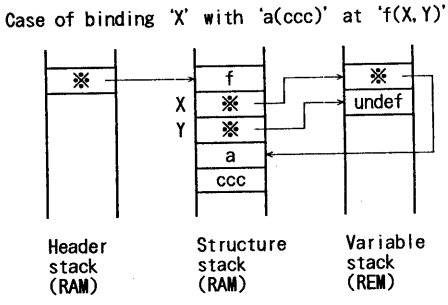


図８　RUM内のスタックの使用例

GIM	RUM	PAM
●ゴール項目列、属性を転送 [→PAM]		●引数間並列での検索開始
節検索失敗の場合（ディープバックトラック）		
●ディープバックトラック処理 制御スタックを以前の状態に戻す RUMのスタックフレームのアドレスを転送 （先頭に戻る） [→RUM]	●以前のスタックフレームにバックトラック	●検索失敗の結果を返す [→GIM] ●既にGIMから書き込まれているディープバック用の中断情報を使って新たな節検索を開始
節検索成功の場合		
●中断情報を制御スタックに格納 ●次候補節検索を指令 [→PAM] ●（単一化成功の場合の）実行ゴールの第1レベル（変数以外）を転送 [→PAM] ●ディープバックトラック用中断情報を転送 [→PAM]	●節番号を項目表のアドレスに変換 ●変数と構造体の単一化開始	●中断情報 [→GIM] 節番号 [→GIM・RUM] ●次候補節検索開始
単一化成功の場合		
●制御スタック更新 （先頭に戻る）	●単一化成功の結果を返す [→GIM・PAM] ●スタックフレームの更新 ●次実行ゴールのインスタンスを格納 [→PAM]	●次候補節検索打ち切り ●インスタンスを受け取り、検索を開始
単一化失敗の場合（シャロウバックトラック）		
●PAMの次候補節出力待ち （先頭に戻る）	●単一化失敗の結果を返す [→GIM]	（次候補節の検索を続行中）

図9　推論マシンの実行の手順

3．2．推論マシンの実行方式

前節で示した構成での処理系の動作について述べる。前に述べたように、この推論マシンでは機能メモリを使った高速化に加えて、処理系を同時実行可能な2つのモジュールと PAMに分け、これらを機能分散で動作させることによる高速化も考慮している。

この処理系での推論の実行局面は、単一化が成功している場合、単一化が失敗した場合（シャロウバックトラック）そして節検索が失敗した場合（ディープバックトラック）の3つに分けられる。

推論の手順を図9に示す。推論は、GIMからPAMに最初のゴールを与えて単一化候補節を検索することから始まる。

単一化が成功し続ける場合の処理は、次のようになる。まず候補節が見つかると、PAM は候補節の節番号をRUMとGIMへ、中断情報をGIMに出力する。RUMは節番号を受け取って単一化処理の残りを行い、GIM は中断情報を制御スタックに格納し、PAM へ現在の候補節の次の候補節の検索（シャロウバックトラック用）を行わせる。さらに GIMは、その単一化が成功したときの次実行ゴールと、失敗してディープバックトラックが起こった場合に行う節検索用の中断情報を PAMに書き込んでおく。RUMでの単一化が成功すると、PAMは次候補節の検索をしているが、これを打ち切り、さきほど与えられた次実行ゴールの情報を使って推論を進めてゆく。

RUMでの単一化が失敗した場合には、PAMは既にその候補節の検索終了後に GIMから次候補節の検索指令を受けていて検索途中であるので、これが終ると同時にすぐ次のマッチングが実行できる。（シャロウバックトラック）

PAMでの候補節検索が失敗した場合には、GIMとRUMはスタックのRestoreをする。PAMは、これも前もってGIM から渡された中断情報を用いて以前の検索の続きを行う。（ディープバックトラック）

この実行手順では、RUM で単一化を行っているのと同時に、PAMでは次候補節の検索を開始し、GIMでは単一化が成功したとき、および候補節検索が失敗したときに実行を続ける位置をPAMに格納している。

この実行方式では、候補節の検索と単一化の両方がほとんど成功しているような、ほぼ決定的なプログラムに対しても、通常のインタプリタと比較して、機能メモリを用いた事による高速化の利点が得られる。さらに、あるゴールに対する候補節が多いような大規模なプログラムに対しては、PAM による候補節の絞り込みの効果のため、実質的な単一化回数を減少させる効果がある。それに加え、シャロウバックトラック時は、PAMとRUMがパイプライン的に動作するため、実行時間はPAMとRUMの処理のうち時間がかかる方に依存し、もう一方の実行時間はほとんど無視することができる。PAMとRUMの処理の内、どちらの処理がより多くの時間を必要とするかは、実行するプログラムの性質と PAMやRUMの実現方法に依存する。

4．推論マシンの実現

ここでは、推論マシンの実現について述べる。

REMは、制御部をLSI化し、それにメモリを接続する構成になっている。アドレス幅が32bitであるのでREM内で利用するメモリは最大 4ギガワードまで可能である。また動作クロックはプロセッサと同期の 25MHzで動作し、例えば80386に接続した場合は、WRITEはNon-Waitで、READは1または2クロックのWaitでアクセスが可能である。

PAM は、外部との入出力や内部のハッシュテーブルに対する操作などで複雑な動作を行う。このため初期の実現ではプロセッサをプログラマブルコントローラとして用いていたが、高速化のためプログラマブルゲートアレイ(PGA)を用いて専用回路を構築する。PAMのREADは、一致照合の回数などにより実行速度は異なるが、一般に次のように表される。

T_{RD}：リード時間

T_{HS}：HS内部処理時間

T_{HT}：ハッシュテーブルのリード時間

N_{HT}：ハッシュテーブルの参照回数

T_{AC}：AC内部での処理時間

N_{AC}：AC内部での処理回数

$$T_{RD} = (T_{HS} + T_{HT}) \times N_{HT} + T_{AC} \times N_{AC}$$

ここで、T_{HS}、T_{AC}を100ns、T_{HT}を 50nsとした場合の次のnreverseプログラムに対する読み出し時間を考えてみる。

```
nrev([X|L0]):-nrev(L0,L1),app(L1,X,L).  …①
nrev([],[]).                            …②
app([X|L1],L2,[X|L3]):-app(L1,L2,L3).   …③
app([],L,L).                            …④
```

このとき、①と③の節は、N_{HS}とN_{AC}がそれぞれ1回、②と④の節は、N_{HS}とN_{AC}がそれぞれ2回である。したがって、リード時間T_{RD}は、①と③の節が 250ns、②と④の節が500nsとなる。

RUM は、タグによる多方向分岐機能などを持つ専用のプロセッサにより構築するのが最良である。しかし現在は第一次の試作のため、RISCプロセッサの SPARCをコントローラに用いて実現する。そこで、SPARC を用いたCPU基板を製作した。製作したSPARC基板は、クロック速度10MHzで動作し、1.25Mbyteのメモリ空間をNon-Waitでアクセス可能な仕様となっている。またこの他にデュアルポートメモリ(DPM)が 4Kbyte実装されている。REMはこのRUMのメモリ空間の一部に組み込んで使用される。

GIMも、RUMと同様に専用のプロセッサを用いるのが最良だが、RUMと同様な理由で同じSPARC基板を用いて実現する。

モジュール間の通信は、GIM-RUM 間は DPMで行い、PAMは GIMとRUM両方からのアクセス可能な共有メモリとして実現する。

5．まとめ

この研究報告では、大規模なPrologプログラム向きなインタプリティブな処理系の高速化を目的とする、可逆メモリとパターン連想メモリの二つの機能メモリを用いた推論マシンのアーキテクチャについて述べた。

このアーキテクチャでは、パターン連想メモリにより、全ての引数に対して Clause Indexingされた節を高速に検索することができ、検索の高速化および単一化回数の減少を図った。また可逆メモリをを用いることにより、変数内容に関するバックトラックの高速化・自動化を図った。

また、このアーキテクチャでは処理系を、GIM、RUM、PAM の二つのモジュールと機能メモリに機能分散し、並列に動作させることにより、処理系の処理に内在する並列性も抽出した。

これらの機能メモリの利用やモジュール化に対する効果は、節の数やバックトラックの回数が多いプログラムに対し顕著である。すなわちこの推論マシンは、簡単なリスト処理などのトイプログラムではなく、知識処理や演繹データベースといった実際的で大規模なプログラムの実行に適している。

謝辞

この研究を行うにあたって、多大なご協力を頂いた日立超ＬＳＩエンジニアリング㈱の酒井要部長に深く感謝する。また、日頃ご指導頂いている本学の阿刀田央一教授と五十嵐智助手に深く感謝する。

参考文献

[1] Warren D.H.D.:AN ABSTRACT PROLOG INSTRUCTION SET, Techinical report 309, Artificial Intelligence Center, SRI International, 1983.

[2] 坂井 他:機能メモリに基づくProlog処理系の構成、情報処理学会記号処理研究報告56-5, 1990.

[3] 小原 他:可逆メモリの機能と内部構造、信学技報 CPSY90-89, 1991.

[4] 小原 他:パターン連想メモリの動作と内部構造、信学技報 CPSY90-90, 1991.