高並列推論エンジンＰＩＥについて

後藤　厚宏　，　相田　　仁　，丸山　　勉　，湯原　雅信　，　田中　英彦　，　元岡　　達

（　東京大学　　工学部　）

１．　はじめに

現在、計算機システムの応用分野は、その問題の解決手段があらかじめ明確なものが主であり、計算機システムの開発目標は、与えられた解決手段に従って最小時間で解くことである。

一方、比較的近い将来の計算機システムは知識情報処理の分野に適応する必要があることが指摘されている。知識情報処理では、与えられた問題と知識から問題解決の手段自体を生成しながら問題解決を行なう推論の機能が中心となり、これによって、問題解決の手順が不明確な分野まで計算機システムの応用分野を拡大することが可能となる。

Prolog に代表される論理型プログラム言語を実行するシステムは、述語論理に基づいていることによる言語上の性質のよさ、および宣言的、非決定的、非数値向きといった特徴を持ち、知識システムの初期モデルとして有望である。しかし、上記の特徴は、従来のマシンアーキテクチャになじまないため、処理速度／機能の両面において高性能なマシンアーキテクチャが要求されている。

本報告では、次章において論理型プログラムの実行モデルについて基本的な概念を整理し、その並列処理性について議論する。３章では、論理型プログラムの高度な並列処理と直接実行を目指した高並列推論エンジンＰＩＥ（Parallel Inference Engine ）を提案する。４章では、これまでに検討を進めてきたＰＩＥのハードウェア構造の概要について述べ、５章において研究課題を整理する。

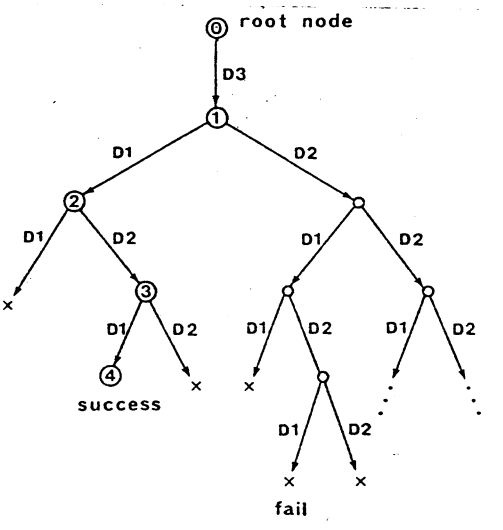
２．　論理型プログラムの並列処理

２．１　証明図によるプログラムの実行表現

Prolog プログラムは、基本的にHorn 節の集合であり、定義節と初期ゴール節から成る。プログラムの実行は、初期ゴール節内のゴールリテラルに定義節の頭部リテラルを適用する（単一化操作）ことによって、次々と新たなゴール節を導出することによって進められ、空節に至った時に成功となる。ただし、ゴールリテラルの実行時に対応する定義節が複数ある場合には、結果として複数のゴール節が導出される可能性がある。そこで、Prolog プログラムの実行は探索木によって表現されることが多い。探索木では、初期ゴール節を根ノードとして、実行途中におけるゴール節が二次元的に配置される。例１のプログラムにおける探索木の例を図１に示す。

```
[D1]  append( nil, *x, *x ) ←.
[D2]  append( (*u.*x), *y, (*u.*z) ) ←
          append( *x, *y, *z ).
[D3]  sublist( *x, *y ) ←
          append( *u, *x, *v ),
          append( *v, *w, *y ).
[G]   ← sublist( (a.*x), (*y.nil) ).
```

［例１］　Prolog プログラム例



［図１］　例１のプログラムを左のリテラルから実行したときの探索木

次に、このようなProlog プログラムの実行を、証明図（Proof　Diagram）を用いて表現することによって、論理型プログラムの並列処理性を考える準備を行なう。

（１）　節のテンプレート表現

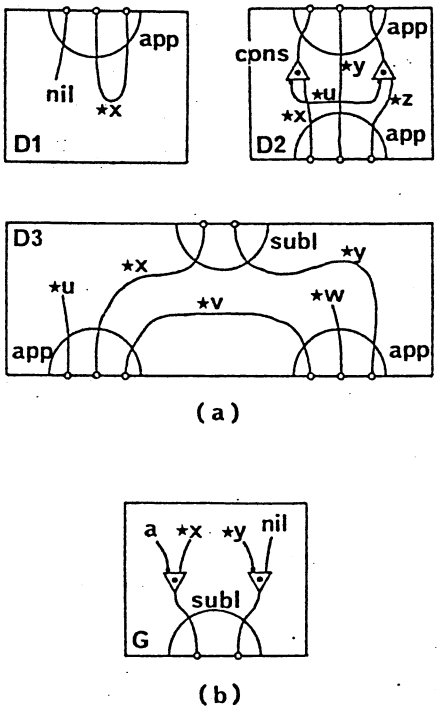
証明図では、例１における各定義節Ｄ１－Ｄ３を図２（ａ）のような定義節テンプレートによって表わし、初期ゴールＧを図２（ｂ）のようなゴールテンプレートによって表わす。ここで、節中の各リテラルは半円によって示され、頭部リテラルは下半円に、ゴールリテラルは上半円に対応する。両半円は、述語記号とその引数に対応した引数ポート（図中の小円）を持つ。変数はリンクで示され、関数式とその引数、定数および引数ポート間を結ぶ。

（２）　ゴールフレーム

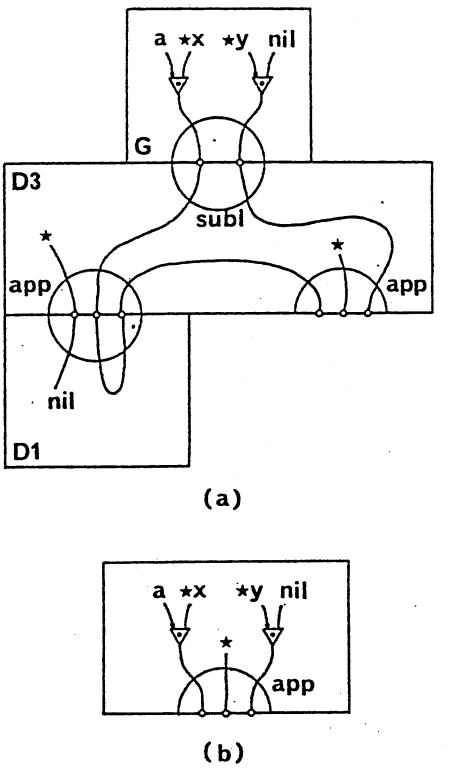
探索木中の各ノード（中間ゴール節）において、根ノードからそのノードまでの計算の中間結果は“中間ゴール節とそれに付随した単一化の環境”である。ここで環境とは、それまでに行なわれた単一化によって生じた変数に対する置換である。この中間結果をゴールフレーム（ goal frame ）と呼ぶことにする。

証明図において、ゴールフレームはテンプレートの組み合わせとして図式的に表現できる。

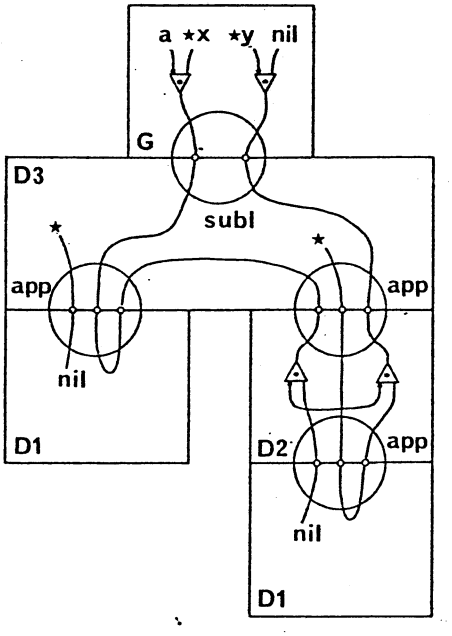
例１において、初期ゴールに対応するゴールフレームは、活性ゴールリテラル“sublist ”を有する。この活性ゴールリテラルと、sublist の定義テンプレートＤ３を接続することにより、新たなゴールフレームが導出される。導出されたゴールフレームには、２つの活性ゴールリテラルがあり、その一方に関して同様な操作を行なうと図３（ａ）のようなゴールフレームが得られる。このゴールフレームは、図１の探索木において根ノードからノード②に至る経路に対応している。ここで、成功した単一化は円によって示され、まだ単一化が必要な活性ゴールリテラルは上半円として残る。



［図２］　節のテンプレート表現



［図３］　ノード②におけるゴールフレーム



［図４］　ノード④における完了フレーム

（３） 完了フレーム

図１のノード④におけるゴールフレームを図４に示す。このゴールフレームは、活性ゴールリテラルを持たず、そのノードに到った計算が成功したことを示している。このような成功ノードにおけるゴールフレームを完了フレーム（ complete frame ）と呼ぶ。この完了フレームは、例１に対する解のひとつを示している。

（４） ゴールフレームの縮退

テンプレートの組み合わせとして表してきたゴールフレーム（図３（ａ））を、同図（ｂ）のように圧縮することをゴールフレームの縮退と呼ぶ。

従来の逐次的なＰrolog の実行では、ゴールフレームの生成におけるやり直しが必要であったため、スタックを利用してテンプレートの組み合わせ方を記憶してきた。しかし、定義節テンプレートの選択肢をすべて一括して適用する処理方式では、やり直しの必要がない。そのため、ゴールフレームを縮退した形で取り扱うことが可能である。

２．２ 問題のＡＮＤ分割とＯＲ分割

論理型プログラムの実行に限らず、一般に大きな問題を解こうとする場合、その問題をいくつかの部分問題に分割し、取り扱いやすくしてからそれらの部分問題を解くことが行なわれる。このとき、問題の分割の方法としては、次の２つの方法がある。

① ＡＮＤ分割

各々の部分問題がすべて解けた場合に初めて全体の問題が解けたことになるもの。

② ＯＲ分割

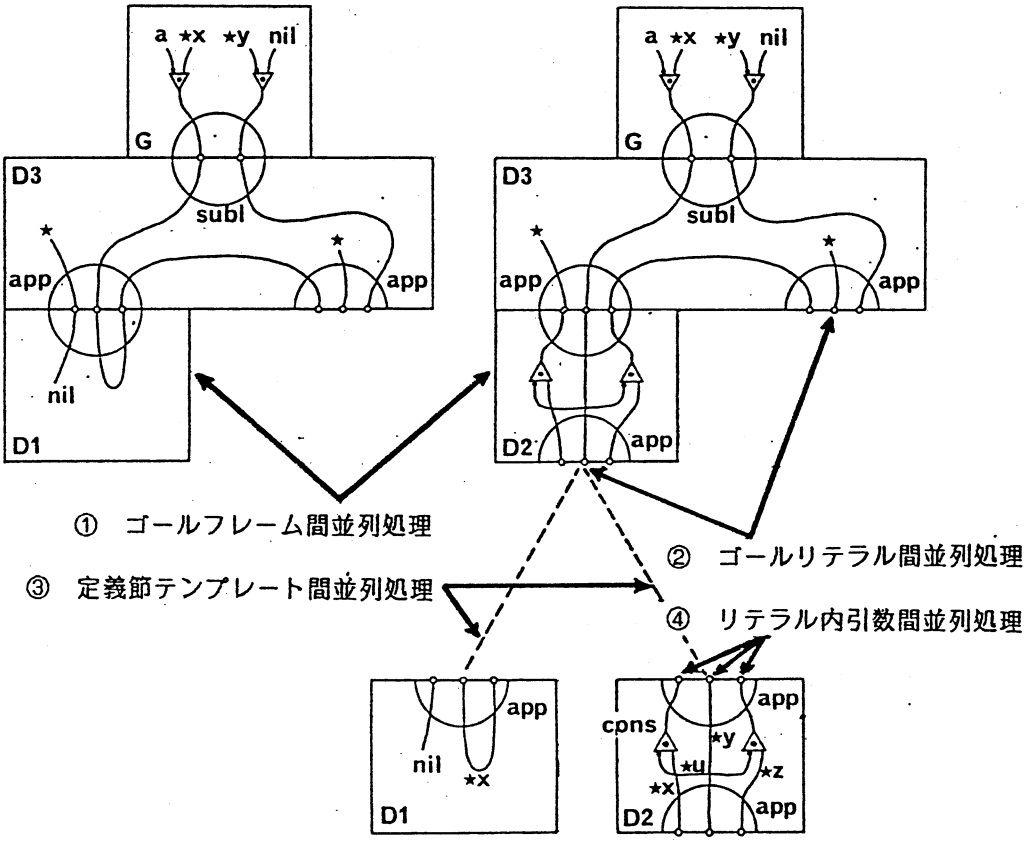
部分問題のうちのいずれかが解ければ、全体の問題が解けたことになるもの。

２．３ 証明図の立場からの並列処理性

論理プログラムの実行を証明図によるゴールフレームと定義節テンプレートの組み合わせとしてとらえた場合、その実行過程において、次の４種類の並列処理を考えることができる。（図５）

① ゴールフレーム間並列処理
② ゴールリテラル間並列処理
③ 定義節テンプレート間並列処理
④ リテラル内引数間並列処理

以下、これらの並列処理の特徴を簡単に述べる。



［図５］ 論理型プログラムの並列処理

（１） ゴールフレーム間並列処理

複数のゴールフレームが処理系内に存在するときに、それらを並列に処理する。

４つの並列処理方法のうちで最も処理レベルの高いものである。これらのゴールフレームがどのようにして生じたかにより実現上の難易度が異なるが、一般に並列処理間の独立性は高い。特に、異なるジョブ間に属するゴールフレームの場合には、互いの影響はほとんど無視できる。

一般にはゴールフレーム間にいろいろな関係が考えられる。ゴールフレームは時々刻々更新されてゆくので、互の関係をゴールフレーム上に記録しておくことは、相手の認識方法の点において難しい。そこで、ゴールフレーム間の関係を表わすための表またはグラフのようなものを別に作っておく。ゴールフレームの更新に従ってその表を更新して、相手との関係を知る手ががりとすれば、一方のゴールフレームの状態によって相手のゴールフレームを消滅／一時不活性化／活性化する等の影響を及し合い、全体として様々な機能を実現することが可能である（３．３節参照）。

（２） ゴールリテラル間並列処理

あるゴールフレームにおいて活性ゴールリテラルが複数ある場合に、それらに対する処理を並列に行なう。

ゴールリテラル間並列処理において、並列に処理するリテラル間に共有される変数がない（それらのリテラルの引数ポート間をつなぐリンクがない）場合には、それらの処理を独立に行なってかまわない。

しかし、それらのリテラル間で共有される変数がある場合には、それらのリテラルを並列に処理した結果が共有変数に関して矛盾をおこしていないかどうかを確認しなければならない。一般に、あるリテラルと照合可能な定義節テンプレートは一通りだけとは限らない。そこでいま、２つのゴールリテラルに対する処理を並列に行ない、それぞれからｍ通り、ｎ通りの処理結果が得られたとする。単純な方法では、ｍ×ｎ通りのすべての組み合せのゴールフレームを作成し、その各々について矛盾が生じていないか検査する必要がある。しかも場合によってはｍまたはｎが無限大になり得るので、この無矛盾性検査（consistency check ）をよほど要領よく行なわないと、並列処理による速度の向上が期待できないばかりか、実行が停止しない可能性もある。

（３） 定義節テンプレート間並列処理

あるゴールフレーム内の特定の活性ゴールリテラルと同じ述語名を持つ定義節テンプレートが複数あるときに、それらとの照合操作を並列に行なう。

問題のＯＲ分割による並列処理であり、論理型プログラムの非決定性に対応する。本並列処理の下では２．１節で述べたゴールフレームの縮退が可能である。

定義節テンプレート間並列処理は、テンプレートとの照合がとれたものについてゴールフレームを複製することにより、ゴールフレーム間並列処理に引き継がれる。

定義節テンプレート間並列処理においては、ゴールリテラル間並列処理で問題になったような変数の値の衝突は起こらない。しかし、この並列処理の下では、並列処理したそれぞれから得られる解が全く同じになったり、一方が他方の特殊解になったりすることがある。この原因としては、

① 定義節テンプレートの形が、もともと、異なる組み合わせ方でも縮退すると同一になるような形をしていた場合

② 初期ゴールフレームの形が特殊な形をしていた場合

の２通りがある。

このような解の重複を検出し、とり除くためには、ｎ通りのテンプレートと照合できた場合に、それらのうちの任意の２個の組み合わせ（ nC2 通り）すべてについて縮退したゴールフレームの照合操作を行なってみる必要がある。これは、ほぼ、ゴールリテラル間並列処理における無矛盾性検査と同程度の負担となる。

しかし、解の重複の検出は省略してもよいことが多い。また、解の重複の原因が上記①の場合には、現在通常のＰrolog 処理系で行なわれているように、カット機能を用いてプログラマの責任において解の重複を防止させることも考えられ、そのような場合には各定義節テンプレートとの照合操作を完全に独立して行なうことができる。

（４） リテラル内引数間並列処理

ある活性リテラルと定義節テンプレートとの照合操作において、複数の引数ポートがある場合に、それらのポートの照合操作を並列に行なう。

この並列処理は、論理型プログラムの実行のレベルの並列処理というよりも、単一化アルゴリズムの並列処理というべきものである。

引数ポート間並列処理も問題のＡＮＤ分割であり、引数間で変数が共有されている場合には、ゴールリテラル間並列処理と類似した問題を有する。しかし、

① 各ポートの照合操作から得られる結果は１通りに限られる、

② ゴールフレーム、定義節テンプレートの両引数がともに構造体であった場合を除き、並列処理が入れ子になることはない、

などの点においてゴールリテラル間並列処理よりも単純である。

本並列処理においては、並列処理を行なう単位が他の3種の並列処理の場合に比べてかなり小さい。このためオーバーヘッドの問題から、並列処理を複数プロセッサに分配して行なうような処理形態には適さないが、ハードウェアとしてテンプレート照合器の中に実装できれば、数倍程度の速度向上が期待できる。

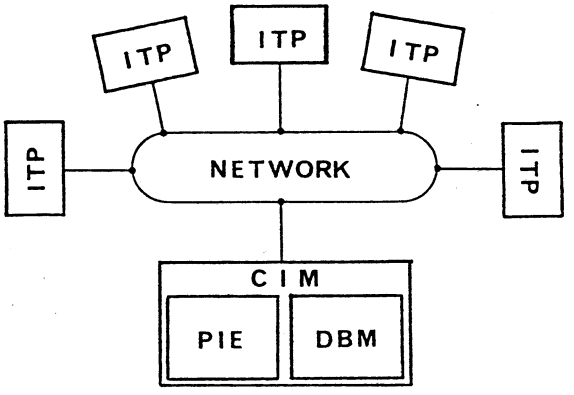
3．　PIEのアーキテクチャ

前章で述べたように、論理型プログラムには様々な並列性があり、それらをどのように実現するかによって様々なマシンアーキテクチャが考えられる。

本章では、現在設計を進めている高並列推論エンジンPIE（Parallel Inference Engine）の第一次モデルの基本アーキテクチャについて述べる。

3．1　PIE第一次モデルの設計方針

PIEを取りまく計算機環境を図6に示す。



［図6］　PIEの計算機環境

PIEは、データベースマシン（DBM）と共に中央推論マシン（CIM）を構成し、複数のユーザに共有される。各ユーザには、高度なユーザインタフェースを提供するITP（Intelligent Terminal Processor）が設けられ、ネットワークを介してCIMを利用する。

このため、PIEにおいて稼動するプログラムは、自然言語処理・自動設計等の応用の中で、データベースから検索された大量のデータを用いた高負荷な部分が中心であると想定される。そこで、PIEのアーキテクチャでは、問題の並列性を充分に生かし、高いスループットを得ることが第一に要求されることになる。

そこで、本章で述べるPIE第一次モデルでは、次の設計方針に基づいてアーキテクチャを考えることにした。

① 定義節テンプレート間並列処理によって高並列性を得る。

② 並列ゴールフレーム間の独立性を高めることによって、①で得られた高並列性を活かしたゴールフレーム間並列処理を実現する。

3．2　PIE第一次モデルの並列処理方式

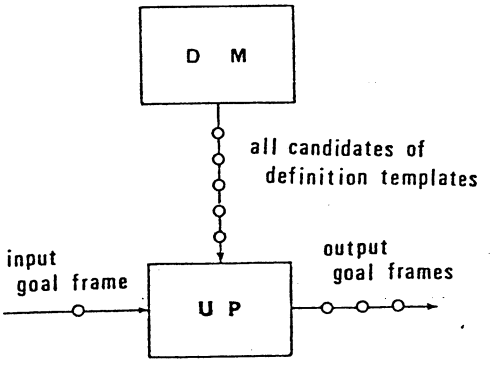
（1）　Unify Processor による並列アクティビティの生成

PIE第一次モデルでは、基本処理要素をユニファイプロセッサ（UP：Unify Processor）と呼ぶ。UPは、ゴールフレームを入力すると、その中のひとつのゴールリテラルについて対応するすべての定義節テンプレートとの単一化を行ない、新たなゴールフレームを（複数）導出する（図7）。ここで、各UPには定義節テンプレートのメモリが付随しており、その各々がすべての種類の定義節テンプレートを持つと仮定している。つまり、本マシンにおける並列アクティビティ（ゴールフレーム）は、主として、各UPが活性ゴールリテラルと、それに対応するすべての定義節テンプレートとの単一化を一括して行なうことによって生成される。

（2）　PIEにおけるゴールフレーム間並列処理

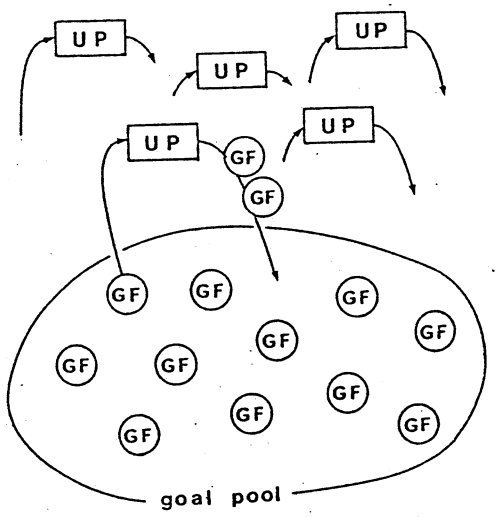
UPによって導出された複数のゴールフレームはゴールプール（goal pool）に蓄えられ、UPに対する新たな入力となる。PIEでは、多数台のUPによって図8に示すゴールフレーム間並列処理が実現される。

PIE第一次モデルでは、このようなゴールフレーム間並列処理の実現を第一目標とし、定義節テンプレート間並列処理および、リテラル引数間並列処理をUPの内部機能として想定する。また、リテラル間並列処理については、実現が容易な部分についてのみ導入し、第二次モデル以降でさらに検討を進

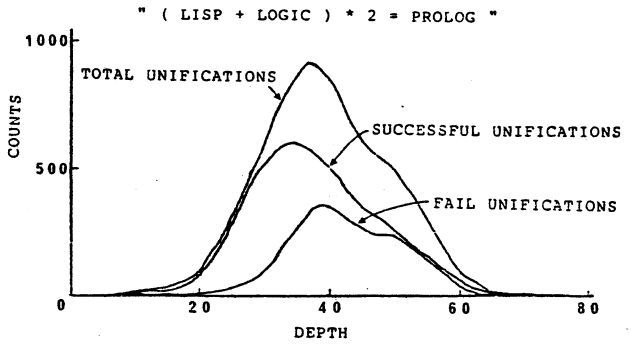


［図7］　ユニファイプロセッサの処理方式

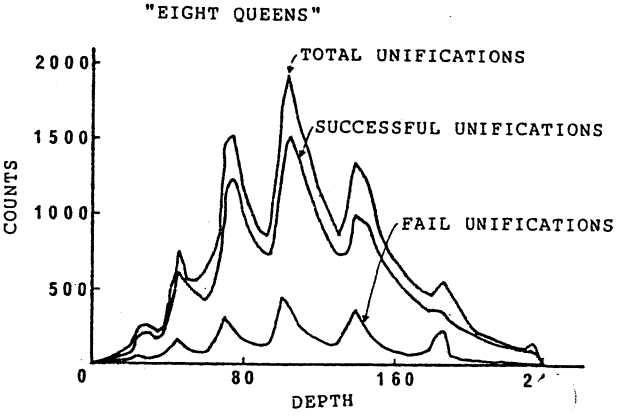
めることにする。以上の設計方針は、このようなゴールフレーム間並列処理に充分大きい並列性があることが判明しているためである。例えば、覆面算“(lisp+logic)×2=prolog”とEightQueensを横型探索により実行した時の並列度を図9および図10に示す。



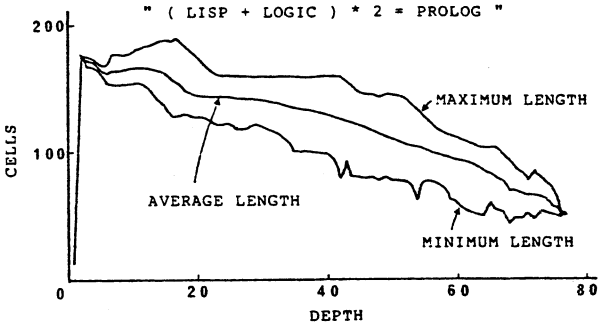
［図8］ ＰＩＥ第一次モデルの並列処理方式



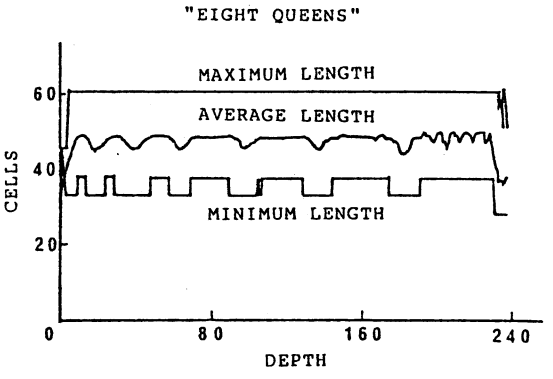
［図9］ 覆面算プログラムの並列度



［図10］ Ｅight Queens プログラムの並列度



［図１１］ 覆面算プログラムにおける縮退の効果



［図12］ Ｅight Queens プログラムにおける縮退の効果

（３）ゴールフレームの独立性

ゴールプールと各ＵＰとの間で転送されるゴールフレーム内には、一般に複数の活性ゴールリテラルが含まれている。一方、ＵＰにおける導出の対象となる活性ゴールリテラルは原則としてひとつである。

ここで、ゴールリテラル間並列処理で問題になったように、ゴールフレーム内の各リテラルは共有変数を通じて強く依存している。このため、ＵＰにおける導出に必要な環境のみを切り出すことは難しい。もし、環境を切り出したとしても、ＵＰにおける導出結果にしたがって、残りの環境に対する操作が必要であり、これがオーバーヘッドとなる可能性がある。また、ＵＰにおける、導出の結果、複数のゴールフレーム（の一部）が生成された場合には、その各々が、残りの環境を共有しあうことになる。これは、共有している部分に対して操作が集中するため、並列性が活かされない危険性を持つ。

そこでＰＩＥでは、ゴールフレーム全体をＵＰとゴールプール間で送受することにより、各々のゴールフレームの独立性を高めることにした。もし、リテラル間の依存関係が弱い時は、リテラル間並列処理によってゴールフレームを分割処理すればよいと考えている。

（４）ゴールフレームの縮退

ゴールフレーム間の独立性を高める方式において、ＰＩＥでは、記憶コストおよび転送コストを押えるために、２．１（４）で述べたゴールフレームの縮退操作を導入する。ＵＰ内では、ゴールリテラルに対応するすべての定義節テンプレートを適用するため、後戻り（backtrack ）がない。これにより、通常の逐次システムにおいて後戻りを考慮してスタック上に残しておく情報の大部分は不要となる。縮退操作は、導出された新ゴールフレームからこのような不要な情報を取り除き、ゴールフレームの大きさを必要最小限にする。

前述（２）で並列性のシミュレーションを行なった２つの例題について、導出時にゴールフレームの縮退を行った時、ゴールフレームのデータ量が導出ステップに従って、どの程度変化するかを評価した結果を図11、図12に示す。ゴールフレームのデータ構造の詳細は省略するが、ゴールフレームを構成している各セルの大きさは、通常の lisp 処理系の１セル程度である。本シミュレーションでは例題数も少なく、縮退を行なわない場合との比較がないため、結論を下すことはむずかしいが、現時点までの結果からは、ゴールフレームの縮退によって、各ゴールフレームのデータ量を小さくまとめることが可能であると思われる。

３．３　ＰＩＥ第一次モデルの制御方式

（１）　ゴールフレーム間並列処理におけるアクティビティ制御

ＰＩＥ第一次モデルにおける並列処理方式は、問題のＯＲ分割に対応しており、各ゴールフレームは原則としてＯＲの関係を持っている。また、現実的な問題解決プログラムの記述ではHorn 節を越えた言語上の制御構造の導入が必要とされており、各方面で言語の拡張が検討されている。このような制御構造の導入はゴールフレーム間にＯＲ以外の依存関係を設けることに他ならない。

一方、３．２（２）で示したプログラムの高並列性は“数の爆発”の問題を引き起こす可能性がある。つまり、多数の処理要素（ＵＰ）を用意したとしても、ゴールプール中のゴールフレーム数は容易にＵＰ数を上回ると予想され、種々の解の探索ストラテジの導入および、資源の枯渇を防止する対策が必要である。

以上の要請に従ったＰＩＥのアクティビティ制御をまとめると次のようになる。

a．多数のゴールフレームの中からＵＰにおいて実行するものを選択する。
b．選択されたゴールフレームをシステム内の負荷が均一になるようにそれぞれのＵＰへ割り付ける。
c．割り付けられたゴールフレームの中から単一化を実行するリテラルを選択する。
d．現実的な問題解決プログラムの記述に必要な言語上の拡張機能を実行する。
e．単一化の失敗等によって不要になったデータを消去し資源を回収する。

（２）　探索木情報によるアクティビティ制御

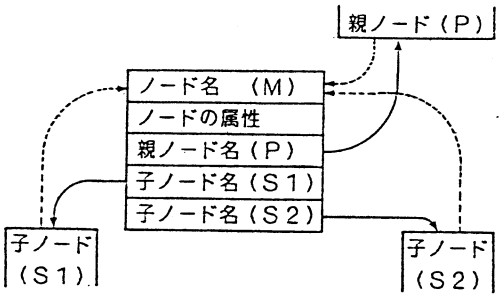
前述のアクティビティ制御において、ａの例としては、探索ストラテジ（プロセッサ数の幅を持った縦型探索／横型探索）に基づく探索深度による制御がある。一方、ｄはif～then～elseに類する制御構造や履歴依存性の記述等、逐次性を導入した実行順序の制御である。この両者を実現する場合は各ゴールフレームの相互関係を明確に把握することが制御機構の基礎として必要である。また、資源の回収ｅの手法として、逐次システムにおける知的後戻りのように失敗経験を積極的に利用する場合も、ゴールフレーム間の関係が重要になる。

論理型言語によるプログラムの計算全体は、探索木として表現できる。並列処理の単位であるゴールフレームを、計算の途中状態として探索木上に位置付けることによって、各々の親子関係を明確にすることができる。また、ゴールフレーム間の並列処理では探索木の各ノードがＯＲノードとなるが、ノードの属性を追加することによって探索ストラテジや言語上の拡張に対応することができる。そこで、ＰＩＥでは、ゴールフレーム間の関係を把握するのに必要な探索木情報を、ノードに新たな属性を導入することによって保持し、さらにノード制御コマンドを設定し、これをノード間で送受することによって、基本的なアクティビティ制御を実現する。

・探索木のノード情報とコマンド

実行途中の探索木は、根ノード・中継ノード（原則としてＯＲノード）・末端ノード（成功／失敗ノード）・ゴールノード（ゴールフレームに対応するノード）によって構成されていると考えてよい。

ＰＩＥでは各ノード操作を並列に実行できるように図13のようなノード情報によって探索木情報を保持する。



ノード名＝＜ノードを保持するＡＣ名：識別名＞
ノードの属性＝＜基本的にはＯＲ＞

［図13］　探索木を構成するノード情報

以下において述べるノード制御コマンドは、
＜宛先ノード名、コマンド名、引数…＞
という型式を持つとする。

・ゴールノードにおける探索木の伸長

ゴールフレームに対応したゴールノードはＵＰによって実行が進められ、新たなゴールフレームが生成される。この一連の操作はＵＰとＡＣ間で送受される以下のメッセージによって表現できる。（図14）

①　resolve　（ノード名、ゴールフレーム）
②　end-resolve　（ノード名、生成ゴール数）
③　newgf　（親ノード名、生成ゴールフレーム）

以上のような探索木の伸長の操作では、コマンド
＜M，son，S＞
を新ゴールノードＳから旧ゴールノードＭへ送ることによってＭ－Ｓ間に親子関係を結ぶ。旧ゴールノードは、すべての子ノードとの関係を結ぶと中継ノードとなる。ＵＰによる実行がすべて失敗した時は末端ノード（失敗ノード）となる。

・末端ノードにおける探索木の刈込み

末端ノード（特に失敗ノード）の多くは探索木情報として不要となり、資源の節約のために刈込まれる。失敗ノードＦは親ノードＰに対して、コマンド
＜P，fail，F＞
を送り自分は（ゴールフレームの実体を含めたノード情報）消滅する。＜fail＞を受けとった親ノードは対応する子ノードをノード情報から削除する。すべての子ノードから＜fail＞が来た場合は失敗ノードとなる。

・中継ノードにおける不要ノードの削除

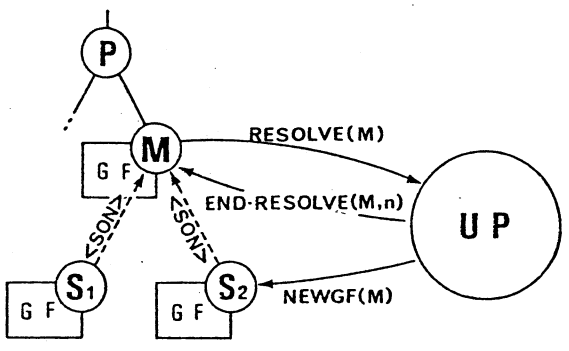
子ノードがただひとつになった中継ノードは多くの場合不要ノードとして削除が可能である。不要中継ノードの削除はノード間の高速なコマンド伝搬を行なう上で重要であり、また再帰を用いたループ等では無数のノードが生成されるため、資源の枯渇を避けるためにも必要となる。

ノード間のコマンド授受が非同期的に実行される環境のため、以下の３種のコマンドとそのノードが削除中であることを示す属性［削除中］を導入する。

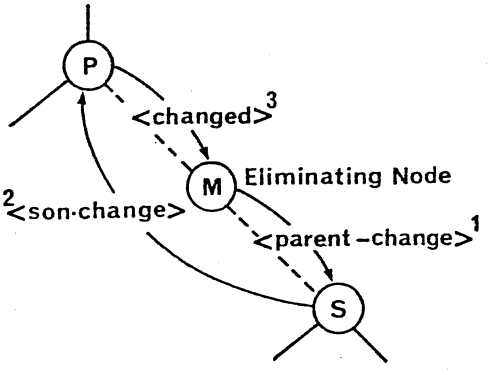
①親がＭからＰに変更されたことを子Ｓに知らせる
＜S，parent－change，M，P＞
②子がＰ２からＭに変更したことをＰ１に知らせる
＜P１，son-change，P２，M＞
③親子の変更操作の終了を削除ノードに知らせる
＜M，changed ＞

不要となった中継ノードは＜parent－change＞を子に送り自分は削除ノードとなる。＜parent－change＞を受けとったノードはノード情報を変更し、新しい親に対して＜son-change＞を送る。　＜son-change＞を受けとったノードはノード情報を変更した後、＜changed ＞を削除ノードへ送る。削除ノードは＜changed ＞を受けとった後、消滅する。

この一連の操作の間、削除ノードは親からのコマンドのうち＜changed ＞以外はすべて子ノードへそのまま渡す。以上の手続きによって非同期的に不要中継ノードの削除が可能である。（図15）



［図14］　ゴールノードの伸長操作



［図15］　不要中継ノードの削除操作

・メタ述語 notの実現例

アクティビティ制御の例としてメタ述語 notを言語に導入する時をとりあげると次のようになる。

not を含むゴールフレーム

← not (goal1), goal2

は、goal1 とgoal2 の間の共有変数がないため図16（ａ）のようにand ノード（ノード属性［ＡＮＤ］）とnot ノード（ノード属性［ＮＯＴ］）を用いた２個のゴールフレームとして考え、goal1 とgoal2 を並列に実行する。

goal1 の実行では、中継ノードがすべてand ノードによって構成される（図１６（ｂ））。

not ノードは、ＵＰにおける導出が空節に至ると失敗ノードとなり、親ノード（and ノード）に対して＜fail＞コマンドを発する。また、導出がすべて失敗した時は成功ノードとなり、＜success ＞コマンドを発する。

and ノードは、すべての子ノードが成功ノードとなった時に自分も成功ノードとなる。一方、子ノードのひとつが失敗ノードとなった時は、子ノードを強制終了させる＜fail-stop ＞コマンドを他の子ノードへ送り、子孫が消滅するのを待ってから自分も失敗ノードとなる。

（３） ゴールフレーム間並列処理の効率化

ゴールフレーム間並列処理における並列度は探索木の横幅に相当し、通常充分に大きい。但し、処理途中におけるすべてのゴールフレームが成功に到るわけではなく、多くのものが失敗の確認に終る。

このため前述（１）のａ，ｃ，ｅのような制御方式によってＰＩＥにおけるゴールフレーム間並列処理を効率よく実現する必要がある。

ａ，ｃ，ｅの各制御には以下のような因子がある。

［Got82］

＜ゴールフレームの選択における因子＞

① 根ノードからの深さ

② 入出力、履歴等によるゴールフレーム間の依存関係

＜単一化を実行するリテラルの選択における因子＞

① リテラル間共有変数によるリテラル間の依存関係

② 手続き呼び出しの決定性

③ リテラルの引数中の変数の数や割合

＜不要なゴールフレームの消去における因子＞

① 導出の失敗／成功

② 失敗原因の分析と追及

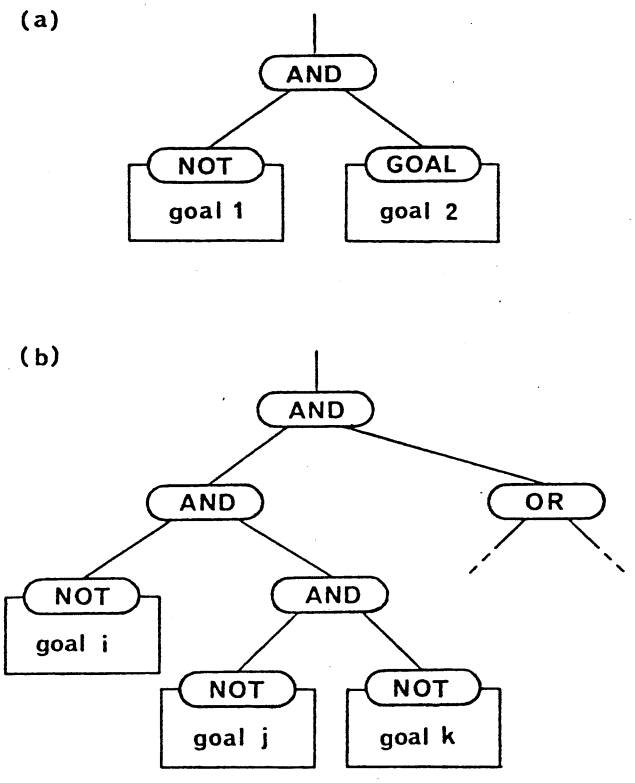
・優先度スコアによる実行制御

上述の制御には多くの因子があるばかりでなく、その制御を指定する側にも、プログラマによって直接に制御が記述される場合と、システムが自動的に判断する場合とがある。これらを統一して扱うために、ＰＩＥでは、それぞれのリテラルやゴールフレームに対して優先度スコアを設ける。優先度スコアは正数のとき優先され負数のとき実行が行なわれないことを示し、また正の優先度スコアを持つもの同士は値の大小が比較される。

・リテラルの選択の最適化における評価

プログラマからの指定がない場合、ゴールフレーム内のリテラルは実行順序の変更が可能である。論理型プログラムにおいてリテラルの実行順序を変えても完了フレームの生成に必要な導出の回数は同じである。しかし、ゴールフレーム間並列処理の環境では実行途中におけるゴールフレーム数が異なる。枝が大きく広がり過ぎないようにして資源の節約を計るためには、ゴールフレーム内のリテラルを選択して実行することが有望である。

現在、変数の数または定数引数や構造体との割合の因子からリテラルの選択を自動的に最適化する手法について、シミュレーション評価を進めている。現時点までに、多くのプログラムが最適に実行しうる優先度スコアの決定法を見い出すことができた。しかし、その優先度スコアの決定法はかなりクリティカルな性質のものであることが判った。このためリテラルやゴールフレームの選択等を単一の優先度スコアにより制御可能であるかどうかについては更に検討が必要である。



［図16］ メタ述語ｎｏｔの実現例

3．4　ＰＩＥの言語機能の基本的方針［Aid83］

３．１節で述べたＰＩＥをとりまく計算機環境下で、ＰＩＥにおいて実行されるプログラムの性格は、

① 会話型処理よりもバッチ方処理が中心

② 大量のデータを用いるものが多い

と考えられる。これらの点を考慮し、ＰＩＥの言語機能としては、次のような特徴を持つものを現在考えている。

① ユーザが入力したプログラムを内部形式に変換する際に、プリプロセッサ、オプティマイザなどによる、ある程度の処理を仮定する。

② 入出力機能などにおいては、会話処理向きなきめ細かい処理よりも、並列性に悪影響を与えないことに重点を置く。

③ マルチユーザ、マルチジョブの環境における問題点を考慮する。

ＰＩＥの機械語については、ハードウェアとのかかわりあいから、まだ流動的な部分も多く、今後、ハードウェア設計の進行にあわせて詳細化してゆく予定である。

4．　ＰＩＥ第一次モデルのハードウェア構造

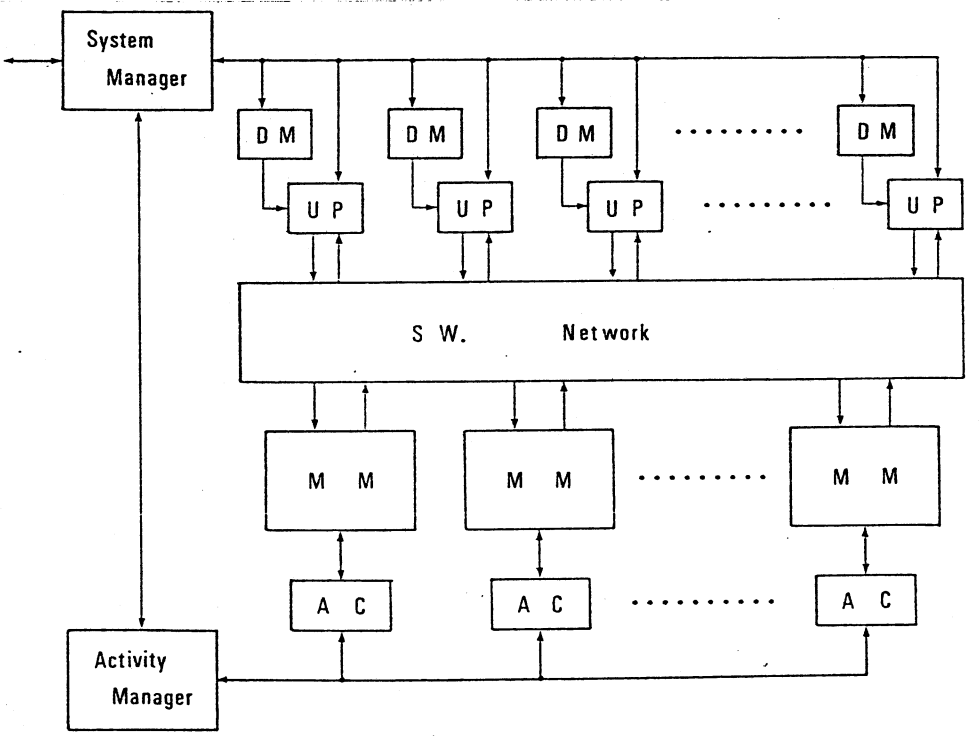
高並列推論エンジンＰＩＥ第一次モデルのハードウェア構成の全体像を図17に示す。［Got83］

4．1　UPとDM

ＵＰ（Unify　Processor）は単一化をハードウェアによって直接サポートする要素プロセッサであり、ＤＭ（Definition Memory ）は定義節を保持するメモリである。ＵＰは、ひとつのゴールフレームを入力し、その中のひとつリテラルについて、対応する定義節のすべての選択肢をＤＭから取り出す。この時、失敗が明白なものについてはＤＭ側でチェックして選択肢を絞り込む。ＵＰは、単一化に成功し新たに生成された複数のゴールフレームを結果として出力する。

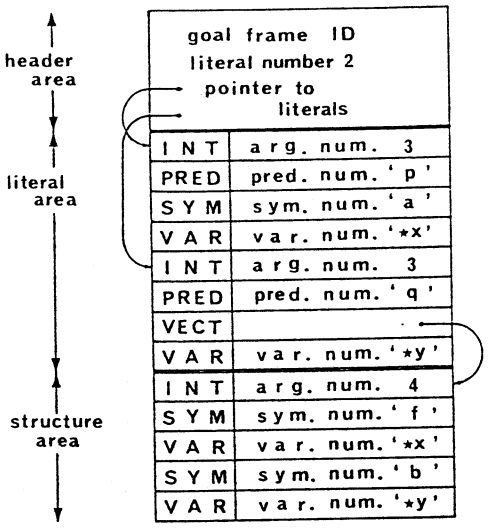
（１）ゴールフレームと定義節の表現

ゴールフレームや定義節は、タグ付の固定長セル（４０ビット程度）が連続した形でＵＰに渡される。図18に、その例を示す。



UP : Unify Processor
DM : Definition Memory
MM : Memory Module
AC : Activity Controller

［図17］　ＰＩＥ第一次モデルのハードウェア構造



［図18］　ゴールフレームのデータ構造

データ型はタグにより識別され、大別すると次の3種類がある。

① アトム

シンボル、整数値などのアトムは、セル1個で表される。シンボルは、シンボル番号で示され、シンボルの印字名は、System Managerが管理するが、実行時には通常アクセスされない。

② 構造体

ベクタ等の構造体は、構造部内へのポインタとして表わされる。

③ 変数

変数は、変数番号で区別する。縮退により、すべての変数は、未定義であることが保証されるので、結合網上やMM（Memory Module ）中でのゴールフレームには、変数の実体を収納する領域は存在しない。

（2）単一化（unification ）

UP内のローカルメモリは、ゴールフレーム側のセルと、定義節側のセルに同時にアクセスできるように、2つに分れている。単一化は、一般に1つのゴールリテラルと複数の定義節の頭部との間で順次行なわれる。そのつど、定義節がDMから定義節側ローカルメモリに取り込まれ、ゴールフレームも入力バッファからゴール側のローカルメモリにコピーされる。

単一化は、ハードウェア化されたUnifierが実行する。

論理変数の束縛状況は、ローカルメモリ中の変数メモリに記録される。変数メモリ内セルのタグは、単一化実行前に、未定義を示す“UNDEF”にされる。単一化の進行に従い、変数の値が束縛されて行く。構造体を単一化を行なう場合には、スタックを用いて再帰的に処理する。

（3）縮退操作

単一化が成功した場合には、新たなゴールフレームを出力バッファに構成し直すが、その際Reducerが縮退操作を行なう。

縮退操作では、参照されなくなったセルを除去し（従来のガーベジ・コレクションに相当）、束縛された変数を、変数を参照しているセルにその内容をコピーした後、取り除く。これにより、“変数のたぐり”がなくなり、次回以降の単一化操作の速度が向上する。同時に、変数番号の付け直しも行なう。

縮退のアルゴリズムは、圧縮型ガーベジ・コレクション方式を応用したものであり、ポインタをたどりながら、必要なセルのコピーを行なう。また、コピー元のセル（又は、別のメモリの同じアドレスのセル）に、コピーしたことを示すフラグと、コピー先のアドレスを残すことにより、ポインタ値の更新も行なう。

（4）リテラルの選択、及び、優先度スコアの計算

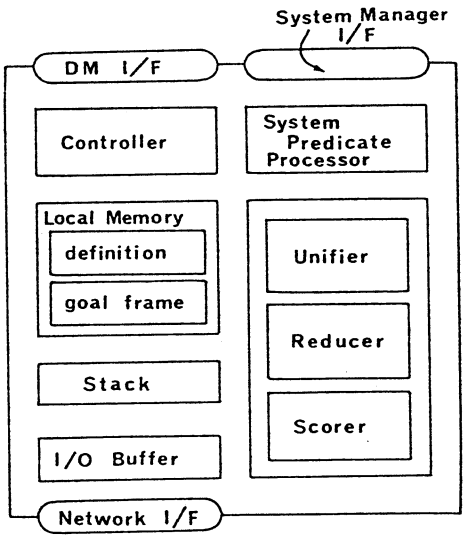
新ゴールフレームから、次回に処理すべきリテラルを選択する。さらに、ゴールフレーム間の優先度を定めるための優先度スコアをゴールフレームの状態から計算する。

（5）新ゴールフレームの送出

新しく作成されたゴールフレームは、結合網を介してMMに分配される。入力ゴールフレームが存在したMMに付随するAC（Activity Controller ）には、新たに生成された子ゴールフレームの数が渡される。

4．2　Memory　Module （MM）

ゴールフレームの実体はMM内に置かれ、全体でゴールプールを構成する。



［図19］　UPのハードウェア・イメージ

各MMには、ハードウエア・ガーベジ・コレクタ設ける。

．３　ACとActivity Manager

１）Activity Controller （ＡＣ）

ＡＣは与えられた問題における探索木の情報を保することによってMM内の各ゴールフレームの関を把握する。また、ＡＣ間の相互通信とその探索情報によって、ＵＰに対するゴールフレームの割付け／効率化のための制御／言語上の拡張機能／源の管理等を行なう。

アクティビティ制御は３．３で述べた他に、子ノドの実行を一時停止するコマンド・実行を再開すコマンド・子孫のゴールノードを実行せずに強制に失敗ノードとするコマンドと、数種のノードの性等を用意することによって実現する。また３．では、各コマンドはノード間で送受信するとしてべたが、実際のシステムではノード情報を管理す多数台のＡＣ間で送受される。

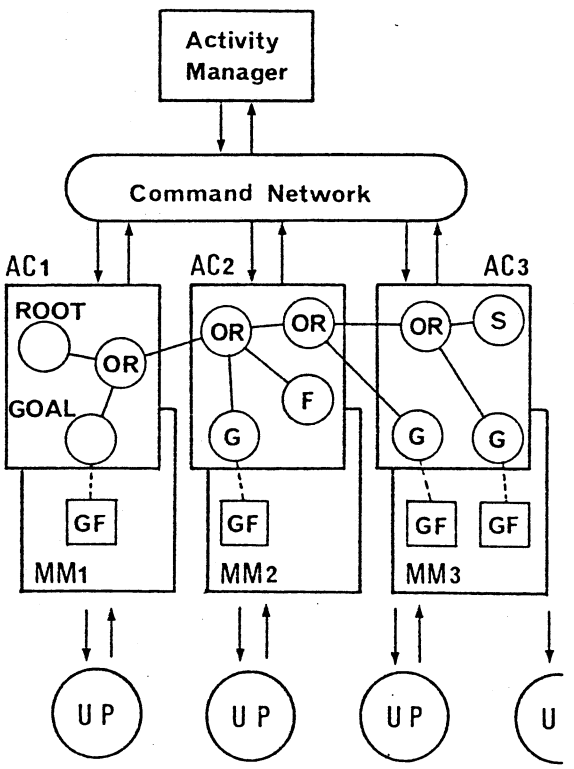
２）Acivity Manager

Activity Managerは、全ＡＣ及びMMを監視し、ステム内の負荷分散を均一にする。

．４　その他

１）System Manager

System Managerは入出力、データベースマシンとの外部インタフェースを司る。



［図20］　アクティビティ制御機構のイメージ

（２）Network

現在、結合網は任意のＵＰ－MM間でゴールフレームが転送できるものを考えている。

５．　おわりに

本報告では、論理型プログラムに関して、まず、実行モデルと並列性について述べ、高並列推論エンジンＰＩＥの基本アーキテクチャについて並列処理方式と制御方式の両面からその基本機能を明らかにした。ついで、現在詳細設計を進めているＰＩＥの第一次モデルのハードウェア構造を示した後、ＰＩＥの基本的な言語機能についても考察を行なった。

現在までに、ＵＰおよびＡＣの基本的な設計が終了し、ソフトウェアシミュレータによりその正当性の検証を急ぐとともに、各種の特性データの収集を行なっている。

今後の課題としては、

① 結合網の設計
② System Manager, Activity Managerの機能の詳細検討
③ ソフトウェアシミュレータによる各種の特性データの収集および整理・検討
④ 言語機能に関する詳細検討ならびにシミュレータへの実装
⑤ ハードウェアシミュレータ（試作機）の設計および製作
⑥ 各種アプリケーションにおけるＰＩＥの性能の測定および評価

といった点があげられる。これらの点については、結果がまとまり次第、報告する予定である。

参考文献

Got82. 後藤　他，“推論向き高並列計算機システムの基本アーキテクチャ”，信学技法　ＥＣ８２－４３（1982）

Got83. 後藤　他，“推論向き高並列計算機システムのアーキテクチャ”，第２６回情処全大，４Ｎ－４（1983）

Mar83. 丸山　他，“推論向き高並列計算機システムのアクティビティ制御機構”，第２６回情処全大，４Ｎ－５（1983）

Yuh83. 湯原　他，“推論向き高並列計算機システムのユニフィケーション制御機構”，第２６回情処全大，４Ｎ－６（1983）

Aid83. 相田　他，“推論向き高並列計算機システムの基本言語機能”，第２６回情処全大，４Ｎ－７（1983）