論　文

高並列推論マシンにおける基本処理単位の構成方式

正員　丸山　勉[†]　　正員　田中　英彦[†]

A Research on Granularity and Representation of Execution Unit in Parallel Inference Machines

Tsutomu MARUYAMA [†] *and* Hidehiko TANAKA [†], *Members*

あらまし　Prologに代表される論理型言語は，近年きわめて重要になりつつある知識処理に適したプログラミング言語である．論理型言語の大きな特徴の一つに手続き型言語のように処理手順を記述するのではなく，その問題を解くために必要な事実・規則のみを記述するという点がある．このため論理型言語はそれに内在する並列性を比較的容意に取り出すことができ並列処理に適していると言える．論理型言語を高並列に実行する並列推論マシンを考えるとき，そのマシンの性能を決定する上で極めて重要な問題の一つに処理単位の構成方式がある．処理単位のレベル・内部表現方式の設定によって，コピー・転送などのオーバヘッドおよび処理単位間の制御などの手間などが大きく異なる．現在までに，幾つかの並列推論マシンが提案されているがそれらのマシンにおいてこの問題が十分に議論されているとは言えない．本論文では，高並列推論マシンにおける処理単位の構成方式について検討およびシミュレーション評価を行う．

1. まえがき

Prolog[(1),(2)]に代表される論理型言語は，近年極めて重要になりつつある知識処理に適したプログラミング言語である．論理型言語は何等かの論理に基礎を置く言語であり，その特徴の一つに手続き型言語のように処理手順を逐一記述するのではなく，その問題を解くために必要な事実，規則のみを記述するという点がある．このため，論理型言語はそれに内在する様々な並列性を比較的容易に取り出すことができ，並列処理に適していると言える．従って，これらの並列性を有効に活かした論理型言語の高速な実行は，知識処理を行う計算機にとって重要な課題の一つである．

論理型言語に内在する並列性としては，

① AND並列性
② OR並列性
③ ストリーム並列性
④ 引数間並列性

の四つがある．これらの並列性のうち，ストリーム並列は基本的にはAND並列に基づくものである．AND関係にある各リテラルが解のproducer，consumerとして動作することによって並列処理を行う（ストリーム並列処理においては，ストリームをOR並列性に基づき複製することは困難であるため，一般には全解探索は行わない）．④は他の並列処理に較べてレベルの低い並列性であり他の処理と組み合せて実行することができる．これらの並列性に基づく並列処理を行うためのモデルとしてはデータフローモデル，書き換えモデル，プロセスモデル等がある[(3)~(10)]．

これらのモデルに基づき上記の並列性を活かした高並列・高速な処理を行う並列推論マシンのアーキテクチャを考えるにあたっては，そこにおいて実行される基本処理単位の設定とその表現方式およびそれらを構成する各データ構造の共有・制御方式の決定がマシンの性能を決定する極めて重要な要素となる[(11),(12)]．本論文では，論理型言語を直接実行するモデルである書き換えモデル，プロセスモデル等に基づく並列推論マシンにおけるゴール表現方式の検討と評価を行う．

2. ゴール表現方式

2.1 並列処理の処理単位

逐次型の処理系においては，論理型言語のOR並列性に基づく非決定性はバックトラックにより実現され

† 東京大学工学部電気工学科，東京都
Faculty of Engineering, The University of Tokyo, Tokyo, 113 Japan

る[13]．逐次型処理系に簡単な変更を加えてOR並列処理を実現しようとするならば，例えばスタックを処理単位と考え並列性による処理の分岐が生じるごとにスタックをすべてコピーし，他のプロセッサに転送するモデルが考えられる．この方法では，処理単位間は完全に独立であるが，一般にはこのスタック長はかなり大きくなる．

これに対して，転送量を抑えるためにスタック全体をコピーするのではなく処理の分岐が起こるごとに，それに対応する環境のみをコピーするというモデルが考えられる．この場合，コピーされた部分によって，それ以前のコピーされなかったデータが共有されるため，共有されている変数に対する値の書き込みによって，共有されている部分のコピー等の操作が必要となる．従って，この方法では共有によって転送量を減らすことができるが，共有されている部分を管理するメモリへのアクセスの集中が問題となる．

以上述べたように，処理単位のレベルをどの程度に設定するかは，一般に処理単位の独立性と処理単位長とのトレードオフとなる．処理単位のレベルを低くすることで処理単位長を抑えることができるが，処理単位間の制御の複雑化および負荷の集中を起こす可能性があり，処理単位のレベルをどのように設定するかは極めて重要な問題である．

並列処理の単位は，主に以下の四つに分類することができる．

① 引数
② リテラル
③ 定義節
④ それ以降の実行に必要なすべての環境

①～④の順に処理単位のレベルは高くなり，④では処理単位間での共有は生じない．①～③においては，複数の引数，複数のリテラル，または複数の定義節をひとまとめとして処理することもでき，これらはそれぞれ②～④と等しくなり得る．しかし，それらは基本的には共有に対する効率化の問題であり，基本的には上記の四つに分類することができる．

上記の①～④に対する処理例を図1～4に示す．これらは単に処理方式の一例であり，他にもいろいろな制御方式を考えることができるが，図1～4は①～④の各処理単位のレベルに応じた処理の特徴を表現している．

図1における処理方式の特徴は，本来AND関係にあるリテラルの引数が並列に処理されるため，各引数の単一化が終了した後，無矛盾性検査が必要である点と(無矛盾性検査を避けるためには動的に引数間のAND関係を判定し引数のクラスタリング等を行う必要がある)，②～④とは異なり各引数の単一化をリテラルという枠に縛られずに実行できる点にある．各リテラルの引数の単一化は各引数の単一化が可能となり次第，他の引数とは無関係に次々と実行される．この方式は，処理のレベルが基本的には1セルであり，また論理型言語は単一代入言語であることから，データフローマシンに適した方式である．

図2の処理方式の特徴は，各ゴールが変数に値を結合した場合に(図2-(D))，その値を親ゴールに書き込む必要がある点と，その時点で定義節中の次のリテラルをゴールとして生成する点にある(図2-(A))．変数に結合された値を親ゴールに書き込むタイミングとしては幾つか考えられるが，ここではそのゴールの実行がすべて終了した時点で親ゴールに値を書き込むものとした．図3の処理方式は図2の処理方式とほぼ同様であるが，処理単位自体が定義節に対応するため，図2のようにリテラルをゴールとして生成する必要がない点(図3-(A))や，親ゴールへの値の結合と子ゴールの生成方法が(図3-(B), (C))多少異なる．②の場合(図2)は③の場合(図3)より処理単位は小さいが，定義節から次のゴールを生成する分だけ処理の手間は大きい．④の場合(図4)では，各処理単位がそれ以降の処理を行うために必要なすべての環境を持っているため親ゴールに値を書き込む必要はなく，② ③とは大きく処理形態が異なる．

2.2 コピーと共有

一般に逐次型計算機における論理型プログラミング言語の処理系においては，リテラルの構造はコピーされず共有の対象となる．これは逐次型処理系においてリテラルの構造のコピーを行っていたのでは，その量が定義節中の変数の数を大きく上まわり，処理速度が低下する為である．並列推論マシンとして専用ハードウェアを考えるならば，これらの定義節中のリテラルのコピーの手間は単一化とのパイプライン実行によってほとんど無視できる程度に押えることができる．この時，処理単位量の増加に伴なう転送のオーバヘッド等はやや増加するが，リテラル構造はコピーしているので，単一化の際にリテラル構造への値の参照が不要，変数領域をリテラル中に埋め込むことにより変数領域の初期化が不要などの利点がある．

並列推論マシンで扱うデータ構造は，

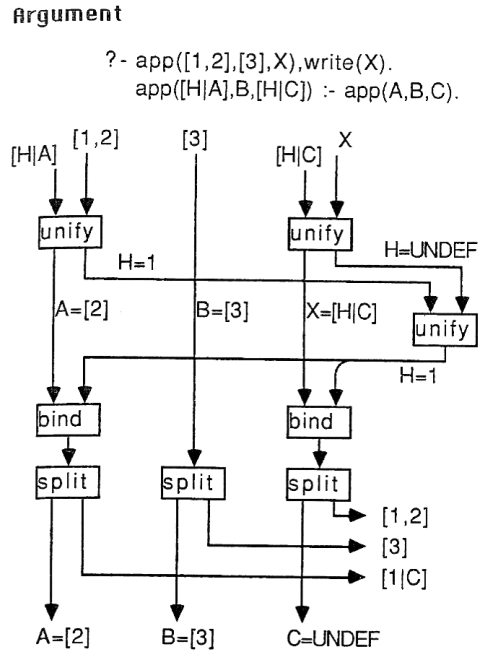


図1　引数レベルでの実行例
Fig. 1　An execution example in Argument level.

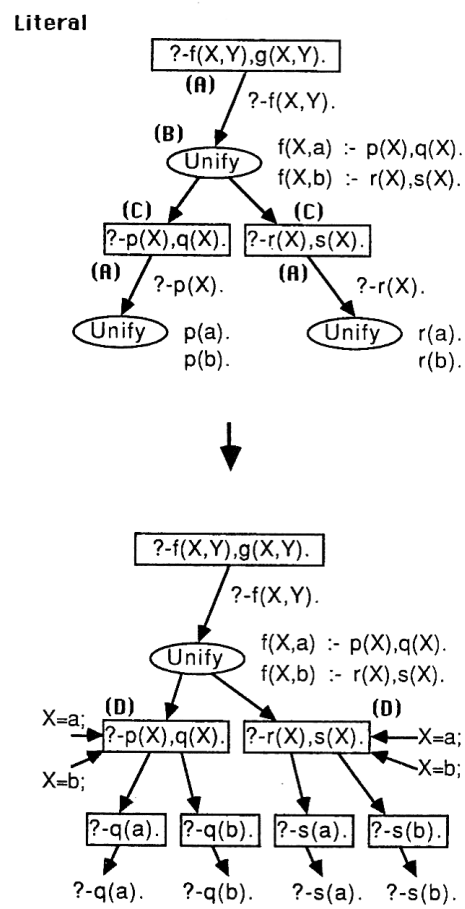


図2　リテラルレベルでの実行例
Fig. 2　An execution example in literal level.

（1）リテラル
（2）構造体 (functor, list)

の二つに大きく分けることができる．従って，これら

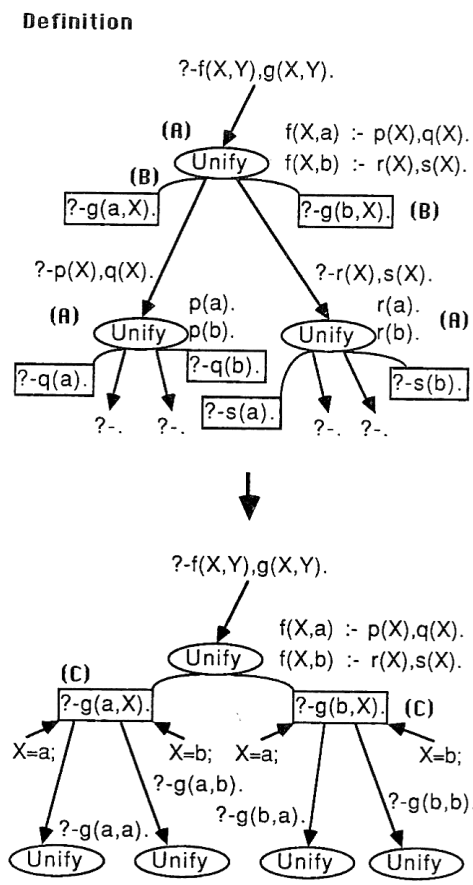


図3　定義節レベルでの実行例
Fig. 3　An execution example in definition level.

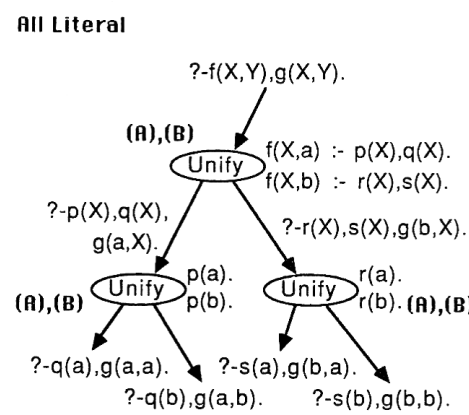


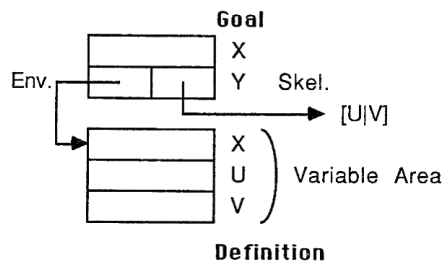
図4　すべてのリテラルレベルでの実行例
Fig. 4　An execution example in all literal level.

の構造をコピーするか，共有するかによって，次の四つの組み合せを考えることができる．

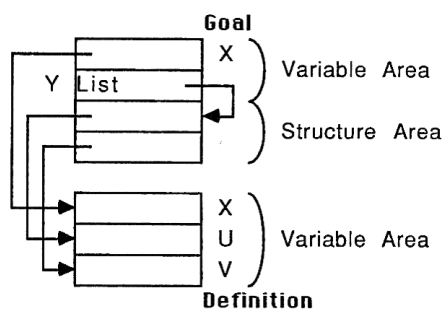
①　リテラル，構造体供に共有
②　リテラルのみ共有，構造体はコピー
③　リテラルはコピー，構造体は共有
④　リテラル，構造体供にコピー

これらのうち，③はリテラルをコピーしても，構造体を共有しているため，構造体へのアクセスがボトルネックとなり，高並列な処理が行えず，リテラルのコピー

に要する時間の分だけオーバヘッドが増すことになり，あまり現実的な方法であるとは考えられないので，ここでは議論の対象とはしない．また，処理単位が引数である場合には各リテラルは前処理においてデータフローグラフ等に展開されていると考えるのが自然であり，リテラルはコピーの対象とはならないので，①および②に含まれるものとする．処理単位が定義節であるとき①，②，④の例を図5に示す．

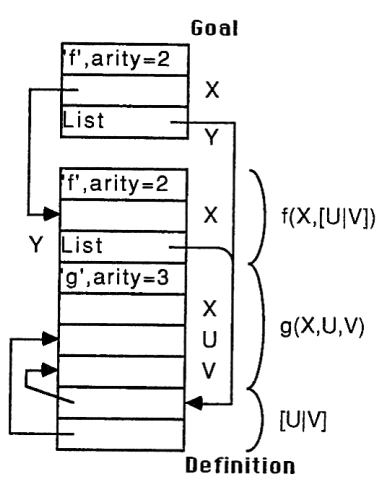


(1) Literal Share / Structure Share



(2) Literal Share / Structure Copy

```
?- f(X,Y).
   f(X,[U|V]) :- g(X,U,V).
```



(3) literal copy / structure copy

図5　ゴールの内部表現
Fig. 5　Internal representaion of goal.

①では変数に構造体が結合された時 molecule (skeleton へのポインタと環境へのポインタの対) として表現される．この方式では基本処理単位としてそれ以降の処理に必要であるすべての環境を持っていない限り単一化の際に親の処理単位への書き込みが起こることになる．この親の処理単位への書き込みを実現する方法として二通り考えられる．一つは，単一化の最中に常に親ゴールの環境を参照しつつ単一化を行い，親ゴールの持つ変数に値が結合された時 molecule の環境へのポインタが親のゴールを指す場合には，親ゴールをコピー (または値が結合された変数のみコピー) するというものである．他の一つは，親ゴールへの参照の手間を避けるためにあらかじめ親の環境のうち必要な部分を自ゴール中にコピーしておき，自分の処理がすべて終了した後で，必要に応じて親ゴールの変数に子ゴールが結合した値を書き込むというものである．このとき親ゴールはコピーされなくてはならない (値が結合された変数のみコピーすることもできる)．しかし，前者においては，親ゴールを管理するメモリへの負荷の集中が問題となり，後者においては構造体の skeleton の持つ変数番号の関係から最小限度必要なデータを親ゴールからコピーすることは難しくそれほど効率的に親ゴールの環境を自分の中に取り込むことはできない．最悪の場合は，それまでの環境をすべて持ち運ぶことになってしまう．

②，④においても①と同じ処理方式が考えられるが，これらの場合には構造体のコピーを行うので①と異なり，skeleton 中の変数番号等の問題は生じず効率的に単一化に必要な最小限度のデータを自分の中に取り込むことができる．②は④より，リテラルの構造をコピーする手間が不要である，また一般にリテラル長より変数の数のほうが少ないため，ゴール長が短くなるという利点を持つ．

3. ゴール構成方式の検討

前節までに，並列処理単位を考える際の主な観点について述べた．これらを組み合せることで，表1に示すような基本処理単位の表現方式を考えることができる．これらの表現方式の優劣を議論する時，評価すべき点として次のようなものを考えることができる．

(a)　基本処理単位長

基本処理単位長は，その処理単位の独立性を損うことがない限りにおいて短い方が良い．

(b)　基本処理単位の独立性

基本処理単位長と密接な関係を持つが，処理単位間の論理的共有の度合は低い程良い．

(c)　単一化の高速性

単一化をより高速に行うことができる処理単位の表現方式がよい．また決定的である定義節に対する処理も高速に行えるものであることが望ましい．

以下，表1に示した表現方式について，それぞれ(a)～(c)の立場から検討を行う．

表1の①，⑤に示した引数間並列処理については二つの立場を考えることができる．一つは，引数間並列性を最大限に活かして高並列な処理を行おうとするものである．この方式においては処理単位のレベルが低いため他の処理方式とは異なり，リテラルの枠に捕らわれずに各リテラルの単一化が可能となり次第，次々とその引数の単一化を行うことができるので，他の処理方式より，より高度な並列処理を実現することが期待できる．しかし，無矛盾性検査（または動的な引数のクラスタリング）や，論理型言語では引数の入出力が動的に決定等の問題があり，その制御はかなり複雑なものとなるため，そのオーバヘッドが問題となる．

他の立場は，引数の並列処理を一つのリテラル内の引数に限定するものである．この場合，引数間並列処理を他の処理方式と組み合せることができる．しかし，一つのリテラルの引数は平均数個程度であり，AND関係にある引数を並列に処理するためのオーバヘッドを考えるならば，それほど有効であるとは思えない．

表1の②，③は構造体のskeletonを共有しているために変数に構造体が結合された場合には，moleculeが作られその環境（処理単位）へのポインタが他の処理単位を指すことがある．既に述べた親ゴールの値を参照しつつ単一化を行い親ゴールの変数に値を結合するという方法については，その親ゴールを管理するメモリへのアクセス競合が起る他，親ゴールをコピーする手間も大きく（値の結合された変数のみコピーする場合には，その値を見つける手間が大きい），その処理がボトルネックとなり，あまり有効な方式であるとは考えられない．また，親ゴールの環境のうち必要なものだけを自分の環境にコピーするという方法も，共有される構造体の変数番号の付け換えが困難であるため（構造をコピーすることも考えられるが，これは表1の⑥，⑦とほぼ同様の方式になる），結局親ゴールのもつ環境をすべてコピーすることになりかねない．従って，⑥，⑦と比較した場合，②，③の方法はあまり効率的であるとは言えない．④のようにそれ以降の処理において，必要なデータをすべて持ち運ぶならば上記のような問題は生じないが，逐次型処理系において実際に使用されるスタック量を考えると非現実的である．以上述べたように構造体のskeletonを共有する方法は，高並列処理には，適してはいないと考えられる．但し，対象とする並列性が数個程度である場合に限れば有効な一つの方法と成り得ると考えられる．

表1の⑥，⑦，⑨，⑩に対しても，②，③の場合と同様に二つの処理方式が考えられるが，親ゴールの値を直接書き換えるという方法は高並列処理には適してはいない．しかし，これらの場合，親ゴール中のデータを自分自身の環境の中に効率的に取り込むことは容易であるため，この方法は有効であると考えられる．

まず⑦と⑩については，⑦の方が，新しく作られたゴールのためにリテラルの構造をコピーする手間が不要である，定義中の変数の数はリテラル長より一般に少ないためゴール長が短くなる，および並列性の無い定義節との単一化に対して処理単位をそのまま逐次処理系におけるスタックの一部とみなすことによってより高速な処理を行うことができる等の利点を持つ．

⑥，⑨と⑦，⑩については，⑥，⑨の方が処理単位が小さい分だけコピー・転送等のオーバヘッドは小さいが，その分リテラル間の分割および変数に結合された値の引き渡し等が必要になり，制御の手間は大きくなる．これら手間は，表2，3に示すように，単一化と同程度の手間となる．どちらが良いかは，ゴール長が大きいことによるコピー・転送等とのオーバヘッドとのトレードオフの問題であり，OR並列処理の場合には，各処理単位がより独立である⑦，⑩が，AND並列

表1　ゴール表現方式の分類

共有／コピー ＼ レベル		引　数	リテラル	定義語	すべてのリテラル
構造体共有	リテラル共有	①	②	③	④
	リテラルコピー				
構造体コピー	リテラル共有	⑤	⑥	⑦	⑧
	リテラルコピー		⑨	⑩	⑪

処理の場合には，解の結合，ゴールの生成などが不可欠であるためよりゴール長の短い⑥，⑨が有利であると思われる．

⑥，⑨については，処理単位のレベルがリテラルであるためゴール長はそれほど変わらないが，ゴールの生成の操作が⑥のほうが容易である（リテラルの構造のコピーが不要）ため，⑥の方が有利であると考えられる．

⑧と⑪については，⑪のほうがリテラルの構造を持つため処理単位長が長いように思われるが，実際には，⑧では単一化を行う時リテラルの構造を参照し，その結果，処理単位中にある変数の値を参照するので，変数番号の問題により不要となったデータをむやみに取り除くことができないため，ゴール長が長くなりやすい．

⑦，⑩と⑧，⑪については，⑧，⑪の方が処理単位長は大きいが，これによるオーバヘッドをパイプライン処理等によって隠すことができるならば，⑧，⑪の方が親ゴールに値を書き込む手間が不要な分だけ有利である．しかし，⑧，⑪では実行プログラムによって，処理単位が長くなりパイプライン処理等によってもそのオーバヘッドを隠すことが不可能になる場合がある．

以上述べた議論からOR並列処理を中心に処理を行う場合には⑦が，

（1） ⑥と較べてゴール長はそれ程大きくならない（ゴール長は，定義節中の変数の数＋変数に結合された構造体のサイズ），

（2） ⑥と較べてゴールの生成など，ゴール間の制御の手間が小さい，

（3） ⑪と較べてゴール長が短い，

等の理由により適当であると考えられる．AND並列処理を行う場合には，処理単位は基本的にはリテラルとなるため⑥が適していると考えられる．

表2 各処理の実行回数とその処理時間〈7queens〉

処理＼レベル	リテラル	定義節	すべてのリテラル
ゴールの生成	19224(30.1)		
単一化(成功)	27284(34.8)	27284(39.2)	27284(26.5)
単一化(失敗)	6589(23.3)	6589(24.8)	6589(21.2)
ゴールのコピー(生成)	27284(23.6)	32763(30.2)	27284(196.0)
部分解の結合	18091(34.4)	4251(45.6)	

表3 各処理の実行回数とその処理時間＜7qa＞

処理＼レベル	リテラル	定義節	すべてのリテラル
ゴールの生成	426(261.9)		
単一化(成功)	599(354.1)	599(344.0)	599(319.4)
単一化(失敗)	3033(79.5)	3033(81.0)	3033(94.3)
ゴールのコピー(生成)	599(174.9)	600(276.8)	599(533.6)
部分解の結合	558(391.1)	40(543.0)	

4. シミュレーション評価

前章で，ゴール構成方式⑦が適当であると考えられることについて述べたが，ここでは，それをシミュレーションによって評価する．構成方式⑥，⑪についても比較のために評価を行うことにする．

構成方式⑥，⑦，⑪の処理手順は具体的には以下のようになる．

（1） リテラル

① リテラルをゴールとして生成する（図2-(A)）

② ユニフィケーションを行う（図2-(B)）

③ ユニフィケーションが成功したものについては，その結果をメモリに格納する（図2-(C)）

④ 必要に応じてユニフィケーションの結果と親ゴールとの結合を行う（図2-(D)）

（2） 定義節

① ユニフィケーションを行う（図3-(A)）

② ユニフィケーションが成功した場合は，その結果をメモリに格納する（図3-(B)）

③ 必要に応じてユニフィケーションの結果と親ゴールとの結合を行う（図3-(C)）

（3） それ以降に必要なすべてのデータ

① ユニフィケーションを行う（図4-(A)）

② ユニフィケーションが成功した場合は，ゴールを生成し，その結果をメモリに格納する（図-(B)）

(3)におけるゴールの生成は，不必要なデータを取り除きゴール長を短縮するために行われる．(1)(2)では不要データの除去は親ゴールとの結合に際して行われる．

図6に示すようなシミュレーションモデルを考えることにする．このモデルは，1台の単一化専用プロセッサと一つのメモリが対となって，推論ユニットを構成し，それらが，ゴール分配網によって多数台結合された形を持つ．従って，このモデルでは，すべての処理を1台のプロセッサが行う．単一化においては，

（1） プロセッサは，メモリを直接参照しながら単一化を行う

（2） 単一化が成功すると，その結果を，他の推論ユニットの負荷状況に応じて，ゴール分配網を用いて他のユニットへ転送，もしくは自分のメモリへコピーする

（3） 単一化の結果ゴールの一部が書き換えられるため，undo stackを用いて元に戻す

という手順で行うものとする．また，構成方式⑥におけるリテラルの切り分け，⑥，⑦における解の結合の処理もすべてこのプロセッサが行い，その結果を各推論ユニットの負荷状況に応じて単一化の場合と同様にゴールの分配を行うものとする．

また，メモリについては，書き込みはプロセッサからのものとゴール分配網からのものがあるが，これらは，同時には行われず，どちらかのみが書き込みを許されるものとする．読出しについては，現在書き込みが行われているゴール以外のものは読出し可能であるとする(これはメモリが多バンク化されている場合にほぼ相当する)．

ゴール分配網[14]は自動的に負荷分散を行うオメガ網である．各プロセッサの負荷状況(具体的には実行可能なゴール数)は，それに接続されるスイッチングユニットに逆向に伝えられ，各スイッチングユニットはその中で最も負荷の軽いものを更に前段のスイッチングユニットに伝える．このようにして，各プロセッサの中で最も負荷の軽いものがどれかがネットワークを逆向きに流れており，ゴールの分配においては，この情報を用いて，ゴールが最も負荷の軽いプロセッサに送られる．また，宛て先を指定したゴールの転送も自由に行うことができる．

シミュレータでは，各処理にかかるステップ数(1ステップを処理するのに必要な時間を1クロックとする)については，既に試作されている試作単一化プロセッサ[15]をもとにして，

(1) メモリへの書き込み，読出しに1ステップ
(2) 各種レジスタの値の退避・復帰に1ステップ
(3) 値の比較・マルチウェイジャンプに1ステップ

等を仮定している．

表2，3に各方式の各処理の回数と，試作単一化プロセッサ程度の専用ハードウェアを仮定した場合の平均処理ステップ数を示す．例題は〈7queens〉と〈7qa〉である．表2，3において解の結合の処理の回数が少ないのは，定義節中の最後のリテラルのユニフィケーションの実行結果を直接の親ゴールではなく，次にユニフィケーションが行われるべきゴールと結合すれば良いからである．この表から解るように，処理単位を定義節とした場合が最も計算量が少ない．

例題〈7queens〉，〈7qa〉に対する稼動プロセッサ数を図7，8に示す．推論ユニットの台数は64台である．例題〈7qa〉はすべてのプロセッサが稼動する状態がそれほど長く続かない場合のものであり，例題〈7queens〉はその状態が長く続く場合のものである．例題〈7queens〉においては，ゴール構成方式⑦が最も良い速度を示し，⑥，⑪と続くことが解る．例題〈7qa〉においてもゴール構成方式⑦が最も良い実行速度を示すのは同様であるが，この場合には，方式⑪のほうが方式⑥より，より良い実行速度を示している．この結果は，表3に示すように，各ゴール構成方式ごとの総処理時間を考えると当然の結果である．

例題〈7qa〉の場合，例題〈7queens〉と較べて特徴的な点は，ゴール構成方式⑥においては，処理の終盤において，数台のプロセッサが比較的長い時間，動作し続けることである．これは，これら数台のプロセッサに負荷の集中が生じていることを示している．これらの場合の各推論ユニット中におけるゴール数の最小値，平均値，最大値を図9，10に示す．これらの図から解るように，例題〈7qa〉においては，ゴール構成方式⑦，⑪においてはゴールが均等に分散されているのに対し，構成方式⑥ではかなりのゴールの集中が生じているこ

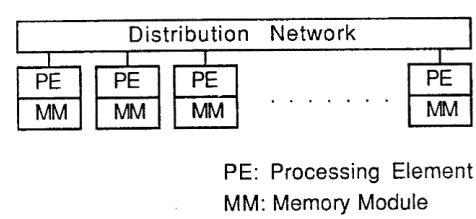


図6 シミュレーションモデル
Fig. 6 Simulation model.

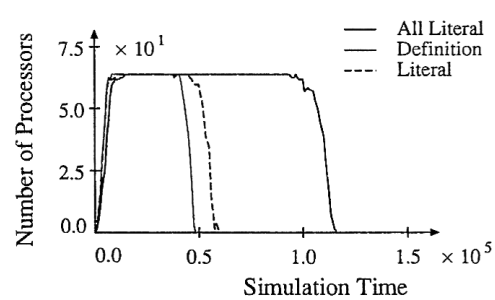


図7 稼動プロセッサ数〈7queens〉
Fig. 7 Number of working processors 〈7queens〉.

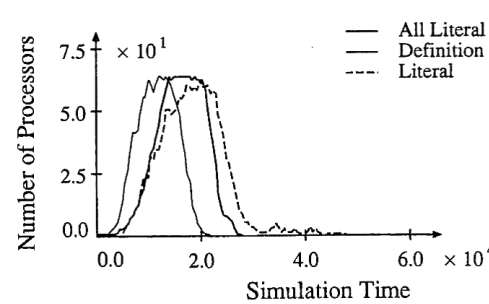
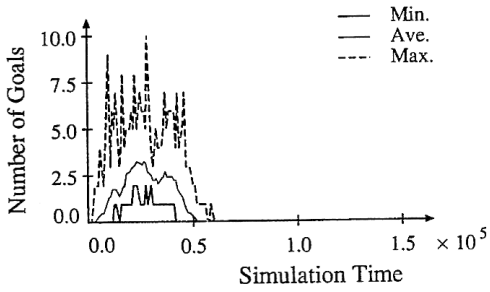
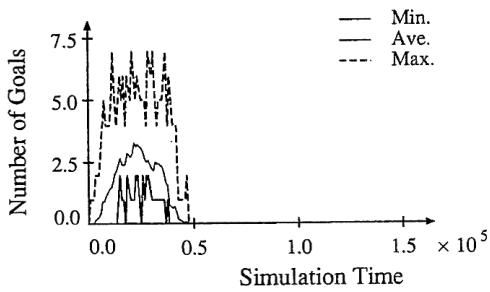


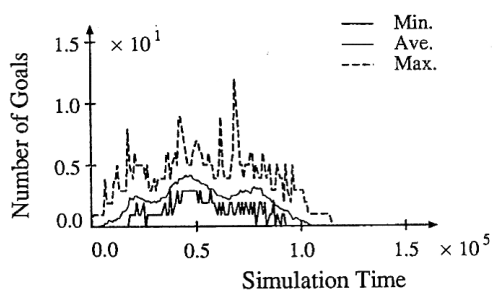
図8 稼動プロセッサ数〈7qa〉
Fig. 8 Number of working processors 〈7qa〉.



Goal Representation - 6

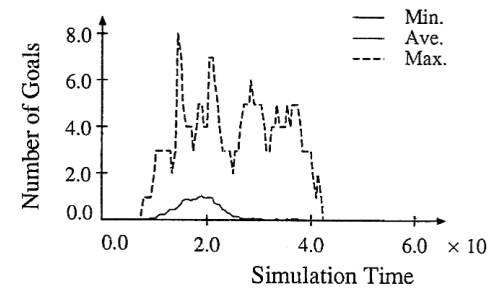


Goal Representation - 7

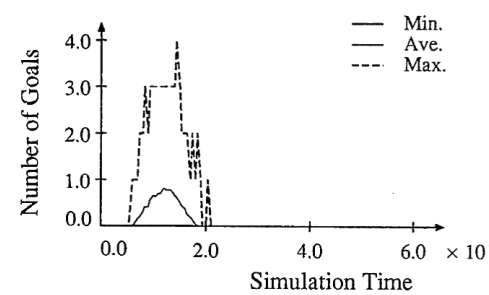


Goal Representation - 11

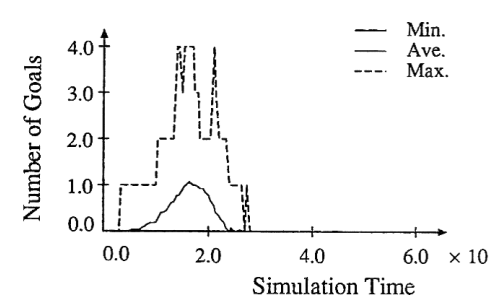
図9　メモリ中のゴール数〈7queens〉
Fig. 9　Number of goals 〈7queens〉.



Goal Representation - 6



Goal Representation - 7



Goal Representation - 11

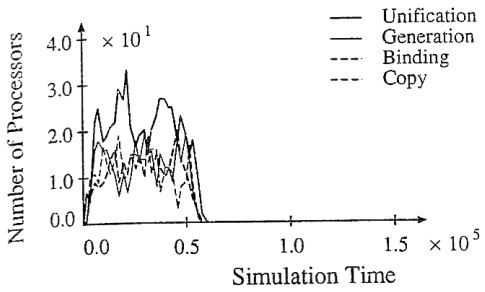
図10　メモリ中のゴール数〈7qa〉
Fig. 10　Number of goals 〈7qa〉.

とが解る．

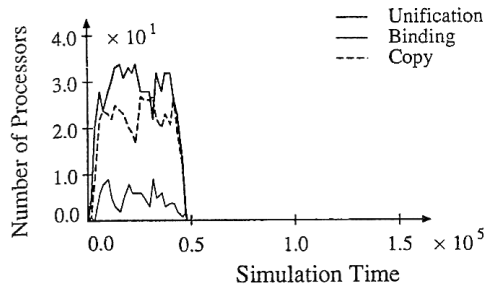
これは，既に述べたように，ゴール構成方式⑪では，ゴール間での共有が全く無いのに対し，構成方式⑦，⑥と共有の度合が大きくなるからである．ゴール間で共有されているゴールはその子ゴールの実行が終了した段階でコピーされるが，このとき子ゴールの実行の終了がほぼ同時に起こると，その結果はOR並列性に基づき，一般に複数個生成されているため，共有されているゴールを管理しているユニットへの負荷の集中が生じる．

構成方式⑦では単一化において，入力ゴールもかならずコピーされるため，負荷の集中が起こるとすれば，入力ゴールに対してではなく，更にその親ゴールに対して起こることになる．しかし，たとえ，子ゴールの実行によって，同一の手間の処理で複数の実行結果が生じるとしても，一般には処理が進むに従って，それらの間で時間差が生じるため，このシミュレーション結果から解るようにそれほど負荷の集中は生じない．

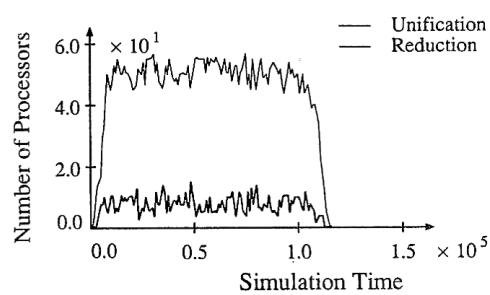
構成方式⑥では，リテラルが切り分けられて，単一化が行われるため，その実行結果は，その切り分けられるもととなったゴールに対して負荷の集中を引き起こす．例題〈7queens〉でもこのような可能性はあるわけであるが，これらの例題においては，切り分けられたリテラルの実行はそれほど簡単には終了せず，そのため，それらの実行結果が切り分けられるもとのゴールに送られるタイミングに時間差が生じ，シミュレーション結果にもみられるようにほとんど負荷の集中を起こしてはいない．しかし，例題〈7qa〉では，切り分けられたリテラルは，常に単位節と単一化されるため，その実行結果がすぐに，切り分けられるもととなったゴールに送られる．このため，そのゴールを管理するユニットに対して，負荷の集中を引き起こすことにな



Goal Representation -6

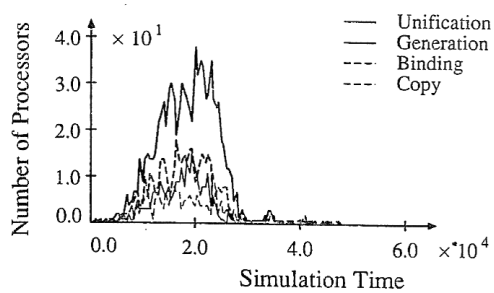


Goal Representation -7

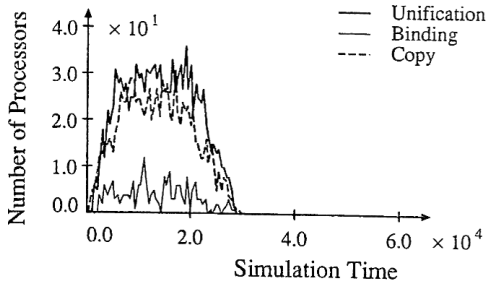


Goal Representation -11

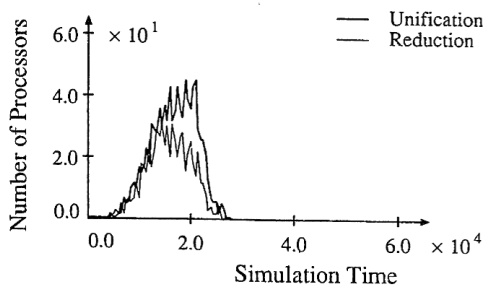
図 11　各処理を実行中のプロセッサ数〈7queens〉
Fig. 11　Number of processors executing each stage 〈7queens〉.



Goal Representation -6



Goal Representation -7



Goal Representation -11

図 12　各処理を実行中のプロセッサ数〈7qa〉
Fig. 12　Number of processors executing each stage 〈7qa〉.

る．

各構成方式における，各推論ユニットの処理の内訳を図 11，12 に示す．まず，例題〈7queens〉の場合についてであるが，この場合には，それほど複雑な構造体の単一化は行われない．ゴール構成方式⑪では，単一化に較べて，ゴール生成を行っているプロセッサの数のほうが著しく多いのが解る．これらの例題において⑪の方式が最も遅いのはこのためである．構成方式⑦では，ほぼ半数のプロセッサが単一化を行っているのが解る．構成方式⑥では，単一化を行っているプロセッサ台数が最も多いものの，それ以上の台数のプロセッサが他の処理を行っている．例題〈7qa〉では，かなり複雑な構造体の単一化が行われるため，例題〈7queens〉に較べて単一化の時間がかなり長くなる．このため，構成方式⑪で半数以上，⑦では 2/3 以上，⑥では半数以上のプロセッサが単一化を行っている．この場合，単一化を行っているプロセッサの台数が最も少ないのは構成方式⑥である．これは，方式⑪では，単一化の時間の増加の分だけ単純に単一化を行っているプロセッサの台数が増えたが，⑥では解の結合などの手間も大きくなるからである（方式⑪におけるゴールの生成は不要なデータの除去を行うためにかなり複雑な操作であるのに対して方式⑥における解の結合は比較的単純な操作である．しかし，この場合のように複雑な構造体を扱うと両者の操作の複雑さは同程度のものとなる）．方式⑦では，解の結合の操作はあまり生じないため，この例題でも最も多くのプロセッサが単一化を行っている．

図 13 に台数効果のグラフを示す．例題〈7queens〉においては，各方式ともプロセッサ台数の増加に対して処理速度が向上していることが解る．このプロセッサ

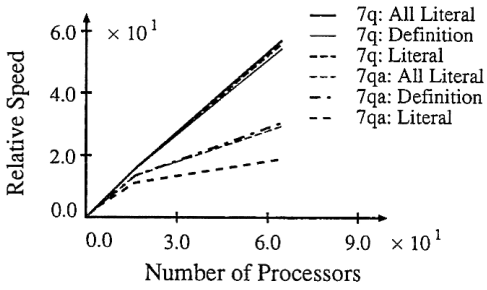


図13 相対速度
Fig. 13 Relative speed.

1台の場合に対する相対速度では，構成方式⑪，⑥，⑦の順でプロセッサ台数の増加に対して，より良い処理速度の向上をみせている．これは，一般に，プロセッサ台数が増加するに従って，総処理時間に対するすべてのプロセッサが稼動状態である時間が短くなるため，相対処理速度は次第に伸びなくなるが，処理速度の遅い方式ほど，他の構成方式の場合よりすべてのプロセッサが稼動状態である時間が長いため，みかけ上の相対処理速度が他の方式より良くなるためである．例題〈7qa〉においては，構成方式⑥の相対処理速度はあまり良くならない．これは，既に述べたようにこれらの例題においては，負荷の集中が生じているためである．

以上の結果より，ゴール構成方式としては，⑦が最も適当であることが確認された．ここではすべての処理を1台のプロセッサで行っているが，各方式において各処理ごとに専用のプロセッサを割り当てパイプライン処理を実現すればより高速な処理を行うことができる．その場合，方式⑥，⑦ではユニフィケーションの手間が他の処理に較べて大きいためユニフィケーションの速度で処理を行う並列推論用プロセッサとなり得るが方式⑪ではゴール生成が処理速度を決める要因となる．しかし，この場合でもパイプラインの段数分だけの並列性がそのプロセッサが最高速度で稼動するために必要なこと，およびパイプラインに必要なハードウェア量を考慮にいれると，台数効果の点などから言って⑦がより優れたゴール構成方式があると考えられる．

5. むすび

高並列推論マシンのアーキテクチャを考えるにあたって，そのマシンにおいて処理される処理単位をどのように設定するかはマシンの性能を決定する上で非常に重要な要素である．本論文では処理単位のレベル，および内部表現方式という観点から処理単位の構成方式について検討およびシミュレーション評価を行った．その結果，

(1) 処理単位のレベルとしては定義節
(2) リテラルの構造は共有，構造データはコピー

という方式がOR並列処理には最も適していると考えられることが解った．AND並列処理においてはできるだけ複数のリテラルを一まとめとする方式が有効であると考えられる．

今後の課題としては，

(1) 上記の処理方式に適した専用プロセッサの検討とそれを用いたより詳細な評価
(2) ストリームAND型言語[16]に適した処理単位および処理方式の検討

等がある．

謝辞 本研究を行うにあたり有益な議論をしていただいたり，貴重なコメントをいただいた高並列推論エンジンPIE研究プロジェクトSIGIEのメンバー諸氏に感謝いたします．

文献

(1) R. Kowalski : "Predicate logic as programming language", Information Processing 74, pp. 569-574 (1974).
(2) W. F. Clocksin and C. S. Mellish : "Programming in Prolog", Springer-Verlag (1984).
(3) A. Goto, H. Tanaka and T. Moto-oka : "Highly Parallel Inference Engine PIE-Goal Rewriting Model and Machine Architecture", New Generation Computing, ICOT, **2**, 1, pp. 37-58 (1984).
(4) T. Moto-oka, H. Tanaka, H. Aida, K. Hirata and T. Maruyama : "The architecture of a parallel inference engine-PIE-", Proc. Int. Conf. FGCS, ICOT, pp. 479-488 (Nov. 1984).
(5) J. S. Conery and D. F. Kilber : "Parallel interpretation of logic programms", Proc. Conf. on FPL and AC, pp. 163-170 (Oct. 1983).
(6) N. Ito, H. Shimizu, M. Kishi, E. Kuno and K. Rokusawa : "The dataflow-based execution mechanism of parallel and concurrent prolog", New Generation Computing, ICOT, **3**, 1, pp. 15-41 (1985).
(7) R. Onai, M. Aso, H. Shimizu, K. Masuda and A. Matsumoto : "Architecture of a reduction based parallel inference machine : PIM-R", ICOT Technical Report : TR-105 (March 1985).
(8) A. Ciepielewski and S. Haridi : "A formall model for OR-parallel execution of logic programs", Information Processing 83, pp. 299-306 (1983).
(9) J. Darlington and M. Reeve : "ALICE : A multiprocessor reduction machine for the parallel evaluation of applicative languages", Proc. Conf. on FPL and CA, pp. 65-76 (Oct. 1981).
(10) J. S. Conery : "The AND/OR process model for parallel interpretation of logic programs", Technical Report :

TR-204, univ. of California, Irvine (June 1983).
(11) 平田，丸山，田中，元岡："高並列推論マシンにおける基本処理単位及び構造記憶に関する考察"，Proc. LPC 85, ICOT, pp. 27-38 (July 1985).
(12) 平田，田中："高並列推論エンジン PIE の構造データの共有方式"，信学論(D)，**J69-D**, 7, pp. 1035-1043 (昭 61-07).
(13) D. H. D. Warren : "Implementing prolog-compiling predicate logic programs Vol 1, 2", D. A. I. Research Report No. 39, 40 (May 1977).
(14) 坂井，小池，田中，元岡："動的負荷分散を行う相互結合網の構成"，情処学論，**27**, 5, pp. 518-524 (昭 61-05).
(15) M. Yuhara, H. Koike, H. Tanaka and T. Moto-oka : "A unify processor pilot machine for PIE", Proc. LPC 84, ICOT, 7-2 (March 1984).
(16) M. Ueda : "Guarded horn cluases", ICOT Technical Report : TR-103 (1985).

（昭和 62 年 1 月 5 日受付）

丸山　勉

昭 57 東大・工・電子卒．昭 59 同大学院情報工学科修士課程了．現在，同大学院博士課程在学中．並列推論マシンのアーキテクチャの研究に従事．情報処理学会会員．

田中　英彦

昭 40 東大・工・電子卒．昭 45 同大学院博士課程了．同年同大講師．以来，ネットワークアーキテクチャ．分散処理．計算機アーキテクチャ，知識処理などの研究に従事．現在，同大助教授．工博．45 年度本会米沢賞受賞．著書「情報通信システム」，「計算機システム技術(共著)」，「計算機アーキテクチャ(共著)」，「ソフトウェア指向アーキテクチャ(共著)」など．