知識ベース指向の並列推論処理システム†

北上 始†† 横田治夫††

人間の知的問題解決を計算機上で実現するためには，知識ベース管理機構と推論機構をあわせもつ知識ベース指向の推論処理システムが必要であると言われているが，知識ベースの大規模化や推論処理における組み合わせ爆発の傾向が強くなるにつれ，システムの高速性が要求される．本論文では，このようなシステムの高速性に答えるために，これを並列化する方法について提案する．基本的な方式は，メタプログラミングの考え方を基礎にしている．本システムの実現するために利用した並列論理型言語 GHC は，単一代入制限をもつプログラミング言語であるので，一つの変数で複数のデータを表現するのが難しい．したがって，GHC により，知識ベース機能を実現するのが極めて困難であることから，並列推論処理を実現するために利用した GHC の変数と，知識ベースを表現するための変数を区別するような，$ 変数を知識ベース側に導入する．これにより，$ 変数を扱う単一化処理や代入処理を実現できる．これらの $ 変数に関連する処理は，GHC で実現しても効果がないことから，これらを逐次のC言語で実現する．また，大規模な知識ベースに対処するために，GHC から並列アクセスが可能な知識ベース検索機構を実現する．最後に，本システムの並列性能を評価するために，本システムを中核とする制約論理プログラミングシステムを作成したので，その結果について報告する．

1. はじめに

人間の知的問題解決を計算機上で実現するためには，知識ベース管理機構と推論機構をあわせもつようなシステムが必要と言われている[1]．このような知識情報処理を目指したシステムは，オブジェクト指向や制約指向のプログラミングパラダイムを吸収しながら，医療分野・電気機械システムの診断や分析をはじめ，電力系統の設備計画・生産計画・輸送計画等，さらには，電気機械システム等の設計や自然現象等のシミュレーションにも利用されるようになってきている．

しかし，知識ベースの大規模化や推論処理における組み合わせ爆発の傾向が強くなるにつれ，システムの高速化が，益々，強く要求されるようになってきている．そのため，並列推論マシン PIM[2),3)] の出現が大変期待されている．特に，遺伝子情報解析[4)]・輸送計画[5)]・LSI の設計[6)]・粒子系のシミュレーション[7)]等では，数時間を遙かに超えるような事例も数多く見受けられるようになってきている．並列推論マシンの利用は，記号処理のサイドに立って，上記の問題を高速に解くための有効な一つの研究アプローチであると考えている．

本論文では，並列推論マシン PIM の実験機として，共有メモリー型のマルチマイクロプロセッサ Symmetry S81 を使用し，知識ベース検索機構を含む並列推論処理システムをメタプログラミング[8),9)]の考え方を基礎にして，試作・評価したので，その結果について述べる．

本システムの基本動作は，人間の知識が登録されている知識ベースに対し，①並列推論を進めるために必要なデータを OR 並列で検索した後，②並列推論の中でそのデータを AND 並列で分析・加工する処理であると，見なしている．しかしながら，従来の記号処理向きの並列プログラミング言語では，これらの二つのタイプの並列処理の研究は，別々に進められていた．例えば，OR 並列に力点を置く研究には，PRISM[10)]，P-Prolog[11)] などがあり，AND 並列に力点を置く研究には，GHC[12),13)]，Parlog[14)]，Concurrent Prolog[15)] などがある．本システムを実現するには，OR 並列と AND 並列の両方を記述する機能が必要であることから，何れの並列プログラミング言語でも難しい．OR 並列処理の部分を AND 並列処理に変換する研究[16)]も精力的に進められてはいるが，完全に変換する方法は未だ得られていない．

本論文では，並列推論マシン PIM の核言語として，GHC が選択されているという事情により，AND 並列言語 GHC に OR 並列の知識ベース検索機構を導入することにより知識ベース指向の並列推論処理を行う方法について述べる．OR 並列の知識ベース検索機構を，AND 並列言語 GHC に導入するために，単一代入制限をもつ GHC の変数と知識ベースを表現する変数を完全に区別する方法を提案する．本システ

† A Knowledge-Based Parallel Inference System by HAJIME KITAKAMI and HARUO YOKOTA (FUJITSU LIMITED).

†† 富士通(株)

ムは拡張性に富むメタプログラミングの方法を基礎にして実現されているので，カットシンボルの処理が自然に実現できることも示す．また，GHC 処理における無駄なメモリー消費を回避するために知識ベースに対する変数処理を GHC で作成することを避け，C言語で実現している．さらに大規模な知識ベースの高速アクセスを実現するために，知識ベース検索機構[17]を GHC から並列にアクセスする方法を示す．具体的には，GHC プロセスと交信可能な知識ベースの検索プロセスを共有メモリー上に実現し，お互いのプロセスのインタフェースを取る方法を示す．検索プロセスは，知識ベースへの検索コマンドごとに用意される．

最後に，本システムを評価するために，本システムを中核とした制約論理プログラミングシステムを作成し，その測定結果について示す．測定では，静電界の問題[18]の例題を扱っている．一般に，制約論理プログラミングシステム[19]は，①知識ベースから制約式を抽出する処理と，②制約式を評価する処理とに分けられるが，ここでは，前者を，本システムにより実現し，後者を，連立一次方程式に帰着されるような制約式に限定し，これを GHC による AND 並列処理により実現している．測定の結果，並列知識ベース検索に1台の PE を割りつけ，GHC 処理に9台の PE を割りつけることにより，GHC 処理に1台の PE を割りつけた場合と比較し，約9倍の並列性能が得られている．

以上により，マルチマイクロプロセッサ Symmetry S81 上で，効率的に並列動作する知識ベース指向の並列推論処理システムが試作されたことを示す．

なお，本研究は，第五世代コンピュータプロジェクトの研究の一環として，行われたものである．

2. メタプログラミングによる推論

並列に動作する本システムの基本的な実現方式は，論理型プログラミング言語 Prolog のメタプログラミングの考え方をヒントにしている．したがって，ここでは，並列処理との関わりに触れながらメタプログラミングによる推論処理方法について簡単に説明する．

通常のプログラミングでは，メタレベルの言語処理系の上で，オブジェクトレベルのプログラミングをしていたが，メタプログラミングは，オブジェクトレベルのプログラムをメタレベルで解釈するようなプログラミング形式を指す．これにより，メタプログラミングでは，メタレベルのプログラミングが自由にできるようになるので，言語処理系に直接手を加えて修正せずとも，言語処理系の機能を拡張したプログラミングが容易に可能になる．

```
ancestor(X, Y):- parent(X, Y).
ancestor(X, Y):- parent(X, Z), ancestor(Z, Y).

parent(f1, f2).
parent(f2, f3).
parent(f3, f4).
```

図 1 知識ベースの例
Fig. 1 An example of knowledge base.

```
solve( true):- !.
solve((P, Q)):- solve(P), solve(Q).
solve( P):- clause(P, Q), solve(Q).
```

図 2 推論処理の例
Fig. 2 An example of inference processing.

知識情報処理システムでは，図1の例で示されるような知識ベースがオブジェクトレベルとして位置づけられ，図2の例で示されるような推論処理がメタレベルとして位置づけられる．

図1の知識ベースは家系図を表すデータであり，1行目および2行目は，先祖の概念を表現するルールである．先祖を表す項 "ancestor (*X*, *Y*)" は，ある人 "*X*" の先祖が "*Y*" であることを指す．先祖は両親の1人を表す述語 "parent (*X*, *Y*)" に基づいて再帰的に定義されるが，両親を表す述語は，ある人 "*X*" の両親の1人が"*Y*"であることを指す．

図2のメタプログラミングにより実現された推論処理プログラムは，最も簡単な Prolog インタプリタであり，探索木上で一つの解答を探索するプログラムである．問題 "?-solve (Goals)." が与えられると，その中のゴール列 "Goals" は，2行目のルールにより単一のゴールに分解される．このゴールの分析を進めるために，3番目のルールの中で，知識ベース検索が行われ，検索結果得られたルールの条件部の分析がさらに進められる．この分析は，条件部の真偽が付くまで行われる．ゴールが，1行目の停止条件を満たす時，真であり，3番目のルールの中で，知識ベース検索の述語 "clause (*P*, *Q*)" の答えが得られない時，偽となる．偽となる場合は，他の可能性を調べるために，探索木上で推論処理のバックトラックが行われる．

図1および図2からも自明なように，メタプログラミングでは，データとしての知識ベースもプログラムとしての推論処理も区別しないので，先祖を定義するために利用されている変数 "*X*", "*Y*", "*Z*" も，推論処理のプログラミングで利用されている変数 "*P*",

"Q" も，通常の Prolog プログラミングで利用される変数と同じタイプの変数として利用される．

次章以降では，以上のメタプログラミングに基づいて，全解探索用の推論処理を並列化する方法について述べる．その中では，知識ベース検索処理の実現方法が重要になる．そこで，知識ベース検索処理を中心に，全解を求める推論処理の手続きを以下のように整理しておく．

(1) 知識ベース検索

ゴール列の中の各ゴールについて，ゴールと単一化が可能なホーン節を知識ベースから検索する．知識ベース検索の結果，答えが見つからない時，処理(4)を行い，答えが見つかった場合は，一つ答えが見つかるごとに処理(2)を行い，答えがなくなった時，処理(4)を行う．

(2) 変数名の付替え

知識ベースで定義されているホーン節と知識ベースから検索されたホーン節は異なるホーン節として扱われるので，これらのホーン節にまたがって変数名の共有が許されない．したがって，検索されたホーン節内の変数に新しい変数名を付け，(3)の処理へ進む．

(3) ホーン節の分析

検索されたホーン節が，条件部を持っていれば，条件部を新たなゴール列とし，(1)の処理へ進む．条件部がなければ，常に，ホーン節が正しいので推論結果の答えを出力し，次の答えを計算するために，(4)の処理へ進む．

(4) バックトラック処理

ゴール変数にバインドされているデータを解除し探索木上で共通の親ノードを持つ他のゴールを(1)で探すことにより，バックトラックを行う．他のゴールがなければ，さらに上位の親ノードをゴールとして，(4)の処理を繰り返す．さらに，上位の親ノードがない場合は，処理を停止する．

3. GHC による並列化

前章から分かるように，全解を求めるための推論処理の中で，(4)の処理を並列化すると，バックトラック処理は，探索木上で，複数の解を並列に探索する処理に置き換えられる．

しかしながら，これらの並列言語 GHC で記述する場合，(3)はメタプログラミングの考え方を基礎にして実現できそうであるが，GHC の単一代入制限により，(1)の処理を実現するのが極めて難しい．すなわち，GHC では，変数に多重値を取ることが許されていないので，一つの変数で知識ベース検索により得られる複数の解答を受け取ることができない．また，(2)の処理を行うためには，変数と定数を区別する機能がなければならないが，GHC には備えられていない．さらに，(4)の並列探索では，複数のパスを並列に探索するために，パスごとに，一つのゴールを用意しなければならない．そのためには，変数を含むゴールをコピーする必要があるが，GHC の単一代入制限により，それができない．

3.1 $ 変数の導入

このように，GHC 変数に関連した問題点を解決するために，GHC の変数と知識ベースのデータに含まれる変数を，分離し，知識ベースのデータに含まれる変数を特殊な定数で表現する．以後，この特殊な変数を $ 変数[20),21)]と呼ぶことにする．

図3に $ 変数を含む知識ベースの表現例を示す．GHC プログラミングにおいて，$ 変数名の区別は，$(1)，$(2)… で行っているが，以下では，簡単のため，これらを $1, $2, $3… で表現する．知識ベース検索の条件指定は，次のように表現される．

```
?-clause_stream (ancestor ($10, $11),
  OutStream).
```

これを実行すると，OutStream には，以下の解答の集合がリストとして返される．

```
[[(ancestor ($12, $13):-
        parent ($12, $13)), Binf1],
 [(ancestor ($14, $15):-
        parent ($14, $16),
        ancestor ($16, $15)), Binf2]]
```

"Binf1" および "Binf2" は，知識ベース検索に伴い単一化処理によって生じた $ 変数のバインド情報である．

図3に示すように，知識ベースのデータを，$ 変数で表現すると，$ 変数は，GHC にとって特殊な定数なので，GHC処理系の単一化処理機能を直接利用することができなくなる．そのために，$ 変数を持つ項間

```
ancestor($1, $2):- parent($1, $2).
ancestor($1, $2):- parent($1, $3), ancestor($3, $2).

parent(f1, f2).
parent(f2, f3).
parent(f3, f4).
```

図 3 知識ベースの例
Fig. 3 An example of knowledge base.

の単一化命令および $ 変数を持つ項に対する代入命令を実現する必要がある．

(1) unify (Term1, Term2, NewTerm, OutputBinf).

"Term1" (項) と "Term2" (項) を単一化し，その結果を "NewTerm" に返す．"OutputBinf" には，単一化の結果得られた $ 変数のバインディング情報を返す．また，単一化処理が終わっても，"Term1" と "Term2" の $ 変数は，書き換わることがない．例えば，ゴールとして，"unify (p(a, $1), p($2, b), Result, OutputBinf)" が与えられると，"Result=p(a, b)" および "OutputBinf=[b($1, b), b($2, a)]" が出力される．

(2) substitute (InputBinf, Term, Result, OutputBinf).

"InputBinf" に示される $ 変数のバインディング情報に基づき，"Term" (項) を書き換える．また，その結果を "Result" に返し，この処理により得られた $ 変数のバインディング情報を，"OutputBinf" に返す．例えば，ゴールとして，"substitute ([b($1, b), b($2, a)], p($1, $3), Result, Binf)" が与えられると，"Result=p(b, $3)" および "OutputBinf=[b($1, b)]" が出力される．

3.2 並列プログラミング

全解を求めるための並列推論処理は，前述の $ 変数を含む項に対する単一化処理や代入処理を利用し，メタプログラミングの考え方を基礎にすることにより実現できる[22]．ここでは，メタプログラミングの考え方を使いどのように並列プログラミングの考え方を使いどのように並列プログラミングを行うかについて説明する．

図2で示した推論処理は，バックトラックにより解答を一つずつ求める処理であるが，本システムの並列推論処理は，複数の解答を並行に求める処理であるので，図2のバックトラックの中の知識ベース検索結果を一つずつ返す述語 "clause (*P*, *Q*)" を拡張し，知識ベース検索により検索されるホーン節集合をストリームとして受け取るようにしなければならない．前節の "clause_stream" により受け取ったホーン節を順次分析していくような並列推論処理は，次のような並列プログラムになる．

```
bagof_solve ([ ], VL, Binf, His, Result):-
  true|
  『Binf で His の内容を書き換えたものを His1 とする』,
  Result=[[VL, Binf, His1]].
bagof_solve ([true], VL, Binf, His, Result):-
  true|
  『Binf で His の内容を書き換えたものを His1 とする』,
  Result=[[VL, Binf, His1]].
bagof_solve ([P|Q], VL, Binf, His, Result):-
  Q≠[ ]|
  bagof_solve ([P],VL, Binf, His, Result1),
  and_solve (Q, Binf, Result1, Result).
bagof_solve ([P], VL, Binf, His, Result):-
  otherwise|
  clause_stream (P, ClauseStream),
  or_solve (ClauseStream, VL, Binf, His, Result).
```

"bagof_solve" の第1引数には，解くべきゴール列が入力される．ここでは，ゴール列をリストで表現している．第2引数には，解かれたゴール列の出力変数がリスト入力される．第3引数には，推論中に $ 変数にバインドされた情報がリスト出力される．第4引数には，推論中に使用されたホーン節が使われた順にリスト出力される．第5引数には，推論結果として得られた複数の解答がストリームとして返される．

第1番目と2番目のプログラムは，図2の第1番目のプログラムに対応する処理である．上記の第3番目と4番目のプログラムは，各々，図2の第2番目と3番目のプログラムに対応する．これらのプログラムは，複数解を並行に探索する処理であるので，知識ベースから検索された複数のホーン節を次々と処理していくプログラミングになっている．特に，第3番目のプログラムは，AND ゴール列を左から順に解いていくプログラムである．この時，AND ゴール間を共有する $ 変数についても，先に処理されたゴールの $ 変数値を他のゴールに伝播させるために，第3番目のプログラムで，変数 "Result1" を "bagof_solve" から "and_solve" に渡している．第3番目のプログラムの "and_solve" は，次のような並列プログラムによって実現される．

```
and_solve (Q, Binf, [[VL, BinfHis, His]|R],
    Result):- true|
  『3.1 節の "substitute" により，ゴール列 "Q" に変数バインド情報 "BinfHis" を代入し，これにより更新されたゴール列を "Q1" として作成する』,
```

```
    bagof_solve (Q1, VL, BinfHis, His, Result1),
    and_solve (Q, Binf, R, Result2),
    merge (Result1, Result2, Result).
and_solve (Q, Binf, [ ], Result):-
    true|Result=[ ].
```

"and_solve" の第1引数には，解くべきゴール列が入力される．第2引数には，推論中に $ 変数にバインドされた情報が入力される．第3引数には，ゴール列 "Q" の直前に処理された AND ゴールの処理結果が入力される．第4引数には，ゴール列 "Q" の推論結果として得られた複数の解答がストリームとして返される．

第1番目のプログラムは，ゴール列 "Q" の直前に処理された AND ゴールの処理結果が一つ以上の解答を持つ時に，起動される．共有する $ 変数へ変数値の伝播を行うために，直前に処理された AND ゴールの処理結果を一つずつ "Q" に反映させ，それらのゴール列を解く．それらの解いた結果をマージし，マージした結果を "Result" に出力する．第2番目のプログラムは，ゴール列 "Q" の直前に処理された AND ゴールの処理結果が一つもない，すなわち，[ ] のとき起動される．AND ゴール列の中のゴールが一つでも [ ] のとき，AND ゴール列の答えが，一つも得られないので，"Result" に [ ] が出力される．

プログラム "or_solve" については，"clause_stream" によって検索された複数のデータを再帰的に1件ずつ処理するプログラムとして次のように実現可能である．

```
or_solve ([[(Head:- Body), Inf]|ClauseList],
    VL, Binf, His, Result):- true|
    『3.1 節の "substitute" により，検索結果得られた $ 変数のバインド情報 Inf で変数リスト VL を書き換え NewVL を作成』,
    append (Inf, Binf, NewBinf),
    append (His, [(Head:- Body)], NewHis),
    bagof_solve (Body, NewVL, NewBinf,
    NewHis, Result1),
or_solve (ClauseList, VL, Binf, His, Result2),
    merge (Result1, Result2, Resut).
or_solve ([ ], VL, Binf, His, Result):-
    true|Result=[ ].
```

第1番目のプログラムは，"clause_stream" によって検索された複数のデータを再帰的に1件ずつ処理するプログラムである．"clause_stream" によって書き換えられた $ 変数は，$ 変数のバインド情報 Inf に返されているので，この情報を変数リスト VL に反映させる．また，検索されたデータの履歴を取るために，データ "(Head:- Body)" を履歴情報 "His" に追加する．データの条件部 "Body" は，"bagof_solve" により分析が進められる．この結果 "Result1" と "ClauseList" の分析結果 "Result2" を，マージし，その結果を "Result" に出力する．第2番目のプログラムは，検索結果データが一つも存在しなかった時か，検索された複数のデータの再帰的な処理が終了した時に，実行される．この時は，"Result" に [ ] が出力される．

3.3 カットシンボルの処理

前述したプログラミングスタイルをさらに押し進めると，カットシンボルの入った知識ベースに対する並列推論処理への拡張も容易に可能になる．

ここでは，カットシンボルは，簡単のため一つのホーン節に一箇所だけ利用されるものと仮定し，条件節 "P, !, Q" を，以下のような "ifthen (P, Q)" で表現する．

```
children ($1, $2):-
    children ($1, [ ], $2).
children ($1, $2, $3):-
    ifthen (parent ($1, $4),
            children ($1, [$4|$2], $3)).
children ($1, $2, $2).
```

この知識ベースの例では，"children ($1, $2)" に対する質問処理で，両親の1人 "$1" に対する子供をすべて見つけ，その結果をリストで "$2" に返すことを期待している．この例で示されるような "ifthen (P, Q)" を処理するための並列プログラムは，前節の "bagof_solve" の第3番目と第4番目との間に，以下のようなプログラムを追加することによって実現される．

```
bagof_solve ([ifthen (P, Q)], VL, Binf, His,
        Result):- true|
        bagof_solve (P, VL, Binf, His, Result1),
        then_part (Q, Binf, Result1, Result).
```

"then_part" は，"P" の分析結果を使って，"ifthen (P, Q)" の "Q" を分析するプログラムであり，次のような並列プログラムとして実現される．

```
then_part (Q, BindInf, [Result1|Result2],
    Result):- true|
    and_solve (Q, Binf, [Result1], Result2),
```

『Result2 が [ ] ならば Result に [fail] を返し [ ] 以外の場合は Result に Result2 の値を返す』.

```
then_part (Q, BindInf, [ ], Result):-
    true|Result=[ ].
```

さらに，前節の "or_solve" のプログラムに，以下のようなプログラムを追加することによって実現される．

```
or_solve ([[(Head:- [ifthen (Cond, Body)]),
  Inf]|ClauseList], VL, Binf. His,
  Result):- true|
  『3.1 節の "substitute" により，検索結果得られた $ 変数のバインド情報 Inf で変数リスト VL を書き換え，NewVL を作成する』,
  append (Inf, Binf, NewBinf),
  append (His, [(Head:- [ifthen (Cond, Body)])], NewHis),
  bagof_solve ([ifthen (Cond, Body)],
    NewVL, NewBinf, NewHis, Result1),
  next_solve (ClauseList, VL, Binf, His,
    Result1, Result).
```

このプログラム内の "bagof_solve" により "ifthen (Cond, Body)" が分析されると，"next_solve" では，"Result1" が [ ] の時，次の "ClauseList" を，"or_solve" で分析し，それ以外の時，"ClauseList" を捨て，枝刈りを行う．

4. 大規模な知識ベース検索

これまでの説明では，知識ベース検索処理 "clause_stream" を含んだ並列推論処理 "bagof_solve" をすべて並列言語 GHC で実現していたが，このアプローチでは，$ 変数の単一化処理 "unify" および代入処理 "substitute" の $ 変数処理に関わる部分が低並列であるにも関わらずメモリーの消費量が大きいことなどの問題がある．特に，大規模な知識ベース処理では，非常に大きな問題となる．また，GHC によるアプローチでは，永久プロセスを作って大規模な知識ベースのデータを効率的に管理・利用するのが難しい．

以上の点から，現在は，知識ベース検索処理 "clause_stream"，$ 変数の単一化処理 "unify"，$ 変数の代入処理 "substitute" 等を，逐次言語のC言語により実現している．GHC からC言語で作成した単一化処理や代入処理を呼び出すことは，容易であるが，知識ベースのデータを複数の GHC プロセスで共有しなければならない．したがって，このデータへのアクセスを管理する知識ベース検索処理 RBU (Retrieval By Unification)[17] および並列言語 GHC の間で，インタフェースをとる必要がある．以下では，GHC とインタフェースをもつ知識ベース検索処理を並列知識ベース検索処理と呼び，その実現方法について簡単に説明する．

図4に並列知識ベース検索処理部のシステム構成を示す．使用した並列マシンが，Sequent 社製の共有メモリー型マルチマクロプロセッサであることから，このシステムは，UNIX ベースの並列 OS (DYNIX) 上で試作されている．知識ベースのデータは，大規模なデータも扱えるようにするために，ハッシュと木構造で構造化し，共有メモリー上に置かれている．共有メモリー上に置かれたデータは，複数の RBU から並列にアクセスされる．各 RBU は，各々，UNIX の1プロセスに割りつけられる．この並列アクセスに対する排他制御は，項関係と呼ばれるデータの集合ごとに行われ，データ検索かデータ更新かによって，共有モードと排他モードとを使い分けている．GHC からの知識ベース検索は，GHC プロセスと RBU プロセスとの間の連絡により行われる．

このシステムを試作し，検索性能を測定した結果，RBU に6台のプロセッサ，GHC に 10 台のプロセッサを割り当てると，Quintus-Prolog の約 25 倍の性能を得ることが分かっている．また，GHC へ割り当てるプロセッサの台数を増やすことにより，良好な台数

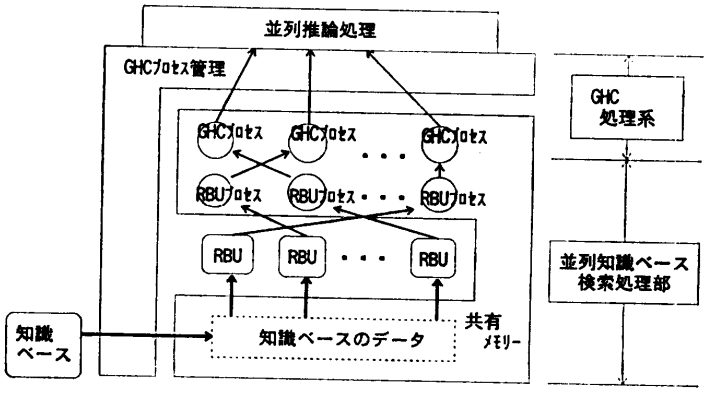

図4 システム構成
Fig. 4 System structure.

効果が得られることも分かっている．

5. システムの評価

本システムは，種々の知識情報処理システムの中核として利用可能である．現在までに，図3に示されるようなデータベースを含む問題[20]や“append”処理等のようなデータベースを含まない問題について，良好な並列性能が得られることを確認している．ここでは，本システムを中核とする制約論理プログラミングシステムの評価を試みる．評価に当たっては，制約論理プログラミングシステムの代表的な例題である静電界の問題を採用する．制約論理プログラミングシステムは，推論中に，知識ベースから数式を抽出できるように，3.2節の“bagof_solve”を拡張し，さらに制約処理系を付け加えることによって，容易に実現することができる．

したがって，本章では，例題としての静電界の問題の説明，および並列推論処理システムを中核とする制約論理プログラミングシステムの測定結果の説明を行う．

5.1 静電界の問題

図5に示される二次元矩形領域において，この領域内のある点の電位がある条件を満足するように，上端の電位を決定する問題について説明する．この領域はメッシュで分割されているが，各格子点の電位は，Liebmann の5点近似法で定められるものとする．

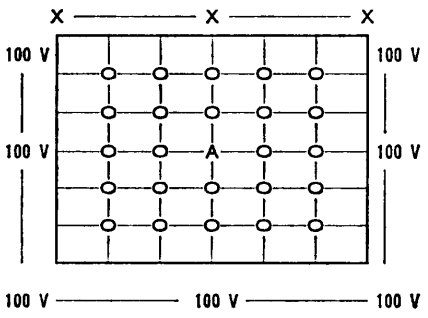

図5 二次元矩形領域
(ただし，$74.9 \leqq A \leqq 75.1$)
Fig. 5 Enclosed 2-dimensional region.

```
design( [$1 | $2] , $3, $4 ):-
    member( $3, [-1, 0, 1] ), laplace( [$1 | $2] ), 74.9 ≤$4, $4≤ 75.1 .

member( $1, [$1 | $2] ).
member( $1, [$2 | $3] ):- member( $1, $3).

laplace( [ $1, $2 ] ).
laplace( [ $1, $2, $3 | $4] ):-
    laplace-vec( $1, $2, $3), laplace( [ $2, $3 | $4] ).

laplace-vec( [ $1, $2 ] , [ $3, $4 ] , [ $5, $6 ] ).
laplace-vec( [ $1, $2, $3 | $4] , [ $5, $6, $7 | $8] , [ $9, $10, $11 | $12 ] ):-
    $10 + $2 + $5 + $7 - 4 ×$6 = 0,
    laplace-vec( [ $2, $3 | $4] , [ $6, $7 | $8] , [ $10, $11 | $12 ] ).
```

図6 静電界の問題を解くための知識ベース
Fig. 6 A knowledge base to analyze electrostatic field.

この問題は，**図6**で表されるような知識ベースとなる．代数式の等号は，簡単のため，$ 変数の単一化命令“unify”で表現することを避け，不等号と同じく，直接，算術記号“=”を使っている．

5.2 測定結果

前節の知識ベースに対して，次のような問い合わせをすると，

```
?-constraint_solve (
  design ( [ [$900, $900, $900, $900, $900],
             [ 100, $911, $912, $913,  100],
             [ 100, $921, $999, $923,  100],
             [ 100, $931, $932, $933,  100],
             [ 100,  100,  100,  100,  100]],
  $900, $999)).
```

最初に，並列推論処理“bagof_solve”が呼び出される．ここでは，図6の“member”の要素ごとに，連立一次方程式を作成する．次に，制約処理系が呼び出され，この方程式が制約処理系に渡される．制約処理系では，これらを行列に変換し，行列計算を行う．行列計算は，**図7**に示すような GHC の AND 並列プログラミングにより容易に実現可能である．

図7では，4×4 行列の例を図示している．各行列要素を GHC プロセスの単位（以下では並列処理単位と呼ぶ）とし，対角要素の上から下へと順に掃き出し法のアルゴリズムを適用していく．

例えば，①の対角要素を起点とする計算は以下のように行う．

（1） 対角要素は，水平方向要素および直垂方向要素へメッセージを送信する．

（2） 水平方向要素および垂直方向要素がメッセージを受け取ると，各要素は，それぞれ，垂直方向要素および水平方向要素にメッセージを送信する．

（3） これにより，垂直方向および水平方向からのメッセージを受け取った各要素は，ただちに，掃き出し法に基づく計算を行う．

並列行列解法の部分の並列性能は，GHC が8台の時，1台に比べて6倍近い並列性能を得ている．

制約処理を結合した並列推論処理システムの測定では，GHC と RBU に合計 15 台のプロセッサを割りつけることにした．この数

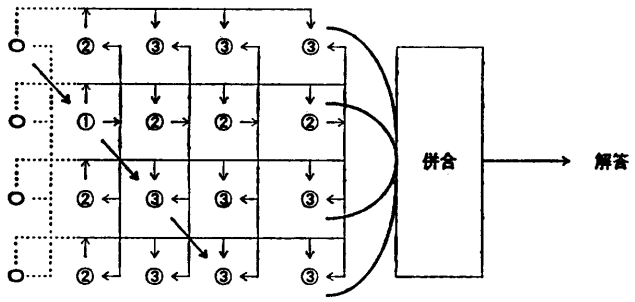

図 7 行列の並列解法
Fig. 7 Parallel solving method of matrix.

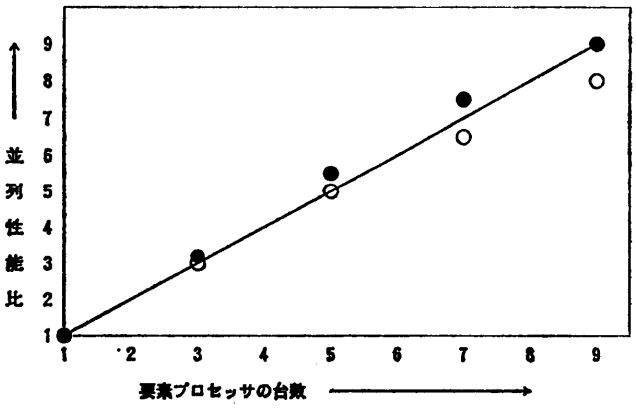

図 8 台数効果の測定結果
Fig. 8 Performance ratio for number of processors.

は，プロセッサ台数を増加した時に，ハードウェアの性能が低下しない数である．ここでは，並列性能比を測定するために，GHC を最大9台まで増加するものとしたため，RBU には，最大6台のプロセッサを割り当てることになった．したがって，RBU に1台または6台のプロセッサを割り当て，GHC に割り当てるプロセッサの台数を 1〜9 台まで増やすことにより，図8に示す結果を得た．図中の●印は，RBU が1台の場合の測定点であり，○印は，6台の場合の測定点である．何れの場合も，プロットされた並列性能比は，GHC が1台の時の処理時間を1として，これに比べて，何倍速くなったかを示している．

○印は，●印に比べ，GHC が1台の時 6% 程度処理速度が速いが，GHC のプロセッサ台数を増加するにつれこの差はなくなる．したがって，○印の並列性能比が，●印に比べて低下したように見える．また，図中の実線では，理想的な並列性能比を表している．以上から，GHC が9台の時，GHC が1台に比べて9倍近い並列性能が得られていることが分かる．

以上の結果から，並列性能が確実に出ている部分は，①並列推論により複数の制約式の組を抽出する部分と，②抽出された複数の制約式の組により複数の行列を作成した後に，各行列を同時計算する部分とにあると考えられる．したがって，この問題では，“member” の要素数が多くなると，制約式を抽出するための OR 並列推論処理が増え，並列度が増加する．また，メッシュの数が増えると，行列のサイズが大きくなるので，これに対する GHC プロセスの数が増加する．これにより，並列度が増加する．

ここでは，等式に帰着される制約問題を扱った．不等式に帰着される制約問題についても，同様な掃き出し法を基本とする単体法のアルゴリズムで解けることから，同様な並列性能が得られると考えている．

6. おわりに

本論文では，並列論理型言語 GHC と逐次のC言語を使い，知識ベース指向の並列推論処理システムをメタプログラミングの考え方を基礎にして実現した．その中で，GHC の単一代入制限を回避するために，知識ベースの変数と GHC の変数を区別した．すなわち，知識ベースの変数を特殊な $ 変数で表し，$ 変数を扱う単一化処理や代入処理を実現した．実現に際しては，メモリー消費量を抑えるために，これらの $ 変数を扱う処理の部分を GHC で実現することを避け，逐次のC言語で実現した．また，本システムを拡張し，カットシンボルが処理できるようにすることも容易であることも示した．さらに，大規模な知識ベース処理にも対処できるようにするために，知識ベース検索機構を実現し，GHC とインタフェースをとった．

最後に，本システムの評価を行うために，本システムを中核とする制約論理プログラミングシステムを作成し，測定を行った．測定においては，静電界の例題を採り上げた．測定の結果，良好な台数効果を得た．

今後の課題としては，部分計算[23),24)]やコンパイル等によるシステム全体の効率化や最適化等が考えられる．

この研究は，今後，膨大な処理時間のかかる各種の応用システムを記号処理的な観点で，並列化し，高速な並列応用を実現するために有用な基本技術を提案している．また，本研究は，リフレクション[25)]とも深い関わりがあり，並列処理にとって，本質的な要素を含んでいる．

謝辞 本研究に当たり，日頃から有益なコメントをくださった，ICOT の KBM 関係の方々および(株)富士通研究所・林人工知能研究部長に深謝いたします．

参考文献

1) 淵 一博：問題解決と推論機構，情報処理，Vol. 19, No. 10, pp. 936-943 (1978).
2) Uchida, S., Taki, K., Nakajima, K., Goto, A. and Chikayama, T.: Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project, *Proc. of the International Conference on Fifth Generation Computer Systems 1988*, pp. 16-36 (1988).
3) Hattori, A., Shinogi, T., Kumon, K. and Goto, A.: PIM/p: A Hierarchical Structured Parallel Inference Machine, *Parallel Computing 89*, pp. 527-532 (1989).
4) 内藤公敏，河合正人，岸野敦子，森山悦子，池尾一穂，伊奈康夫，池坂守夫，佐藤弘幸，五條堀孝：高並列計算機の適用による大量遺伝子情報解析技術の実現，並列処理シンポジウム JSPP '90, pp. 329-336 (1990).
5) 福村 聡，佐能克明，山川栄樹：鋼材出荷計画エキスパートシステムと分枝限定法，オペレーションズ リサーチ，Vol. 33, No. 1, pp. 33-39 (1988).
6) 林 孝雄(訳)：デジタル計算機の自動設計，産業出版 (1973).
7) Sato, H. and Ikesaka, M.: Particle Simulation on a Distributed Memory Highly Parallel Processor, *Supercomputing in Nuclear Applications '90*, pp. 101-102 (1990).
8) Bowen, K. A. and Kowalski, R. A.: *Amalgamating Language and Meta-Language in Logic Programming*, TR 4/81, Syracuse University (1981).
9) Kitakami, H., Kunifuji, S., Miyachi, T. and Furukawa, K.: A Methodology for Implementation of a Knowledge Acquisition System, *Proc. of the 1984 International Symposium on Logic Programming*, pp. 131-142 (1984).
10) Kasif, S., Kohli, M. and Minker, J.: PRISM: A Parallel Inference System for Problem Solving, *Proc. of the Logic Programming Workshop 83*, pp. 123-152 (1983).
11) Yang, R.: P-Prolog: A Parallel Logic Programming Language, *Series in Computer Science*, Vol. 9, World Scientific Publishing Co. Pte. Ltd. (1987).
12) Ueda, K.: Guarded Horn Clauses, Technical Report TR-103, ICOT (1985).
13) 小沢年弘，細井 聡，服部 彰：FGHC 処理システムのメモリ使用特性と世代別ガーベジ・コレクション，情報処理学会論文誌，**Vol. 30, No. 9**, pp. 1182-1188 (1989).
14) Clark, K. L. and Gregory, S.: **Notes on Systems Programming in Parlog**, ***Proc. of the International Conference on Fifth Generation Computer Systems 1984***, pp. 299-306 (1984).
15) Shapiro, E.: System Programming in Concurrent Prolog, *Proc. 11th Annual ACM Symp. on Principles of Programming Languages*, ACM, pp. 93-105 (1984).
16) Takeuchi, A.: 並列問題解決用言語 ANDOR-II，日本ソフトウェア科学会第二回知識プログラムシンポジウム (1986).
17) Yokota, H., Kitakami, H. and Hattori, A.: **Term Indexing for Retrieval by Unification**, *Proc. of 5th Int'l Conf. on Data Engineering*, pp. 313-320 (1989).
18) Heintze, N., Michaylov, S. and Stuckey, P.: CLP(R) and Some Electrical Engineering Problems, *Fourth IEEE Symposium on Logic Programming*, pp. 675-703 (1987).
19) Lassez, C.: Constraint Logic Programming, *Byte Magazine*, August, pp. 171-176 (1987).
20) 北上 始，横田治夫，服部 彰：知識処理向き並列推論エンジン，電子情報通信学会研究会報告，CPSY 88-50, pp. 7-12 (1988).
21) 北上 始，横田治夫，服部 彰：知識処理向き並列推論メカニズム，第 38 回情報処理学会全国大会論文集，pp. 1011-1012 (1989).
22) 北上 始，横田治夫，服部 彰：知識ベース指向の並列推論処理システム，第 41 回情報処理学会全国大会論文集(3)，pp. 23-24 (1990).
23) Furukawa, K., Okumura, A. and Murakami, M.: Unfolding Rules for GHC Programs, *New Generation Computing*, pp. 143-157, Ohmusha, Ltd. (1988).
24) Fujita, H., Okumura, A. and Furukawa, K.: Partial Evaluation of GHC Programs Based on the UR-set with Constraints, *Proc. of the Fifth International Conference and Symposium on Logic Programming*, pp. 924-941 (1988).
25) Tanaka, J., Ohta, Y. and Matono, F.: Overview of an Experimental Reflective Programming System: ExReps, *FUJITSU Scientific and Technical Journal*, Vol. 26, No. 1, pp. 86-97 (1990).

(平成2年7月2日受付)
(平成2年11月13日採録)

北上　始（正会員）

1952 年生．1976 年東北大学大学院修士課程修了．同年富士通(株)入社．1978 年～82 年(株)富士通研究所において関係データベース管理システムの研究・開発に従事．1982年6月～85年5月(財)新世代コンピュータ技術開発機構において知識ベース管理システムの研究に従事．1985年6月(株)富士通研究所に帰属．静止衛星の姿勢制御ソフトウェアの開発，関係データベース管理システムの研究・開発，知識ベース管理システムの研究．情報処理学会 25 周年記念論文に採録．日本認知学会，日本ソフトウェア科学会，人工知能学会各会員．

横田　治夫（正会員）

1957 年生．1980 年東京工業大学工学部電子物理工学科卒業．1982年同大学院情報工学専攻修士課程修了．同年4月，富士通(株)入社．同年6月より(財)新世代コンピュータ技術開発機構に出向．1986年4月(株)富士通研究所に帰属．データベースマシン，演繹データベース，知識ベースシステム，並列記号処理の研究・開発に従事．電子情報通信学会，人工知能学会各会員．