階層型挟み打ち探索による PROLOG OR 並列処理手法†

甲斐宗徳††* 小林和男†† 笠原博徳††

本論文では階層型挟み打ち探索法と呼ぶPROLOGのOR並列処理手法を提案する．本手法では，PROLOGの処理過程をAND逐次実行の条件下でOR木を用いて表現し，そのOR木を複数のプロセッサが左右から階層的に挟み打ちをする形で並列かつ独立に深さ優先探索を行う．これによりプロセッサへの負荷割当て単位（グラニュラリティ）を大きくとることができ，負荷の割当て制御（スケジューリング）の頻度を低減させ，スケジューリングによるオーバヘッドおよび実行時のプロセッサ間データ転送のオーバヘッドを低く抑えることが可能となる．また，スケジューリングの効率化のために各プロセッサの探索状況を示す特殊なポインタ（セレクションポインタ）を導入する．これにより，負荷の割当て後，探索に必要な環境を各プロセッサがデータ転送を行わずに自己生成でき，スケジューリング時のデータ転送オーバヘッドをさらに軽減することができる．本手法ではOR木の左右から深さ優先探索を行うため，m台のプロセッサを用いて1台の時の$1/m$以下の処理時間を得るという加速異常現象を有効に引き出すことができる．本手法の性能および有効性はソフトウェアシミュレーションにより確かめられる．

1. まえがき

PROLOGなどの論理型言語を用いて高速な知識情報処理を行うためには，探索の高速化が重要なポイントとなる．この高速化のために従来より複数のプロセッサを用いた並列処理の研究が多く行われている．これらの研究は，PROLOGの並列処理手法[1)~3)]や並列化論理型言語[4),5)]，およびそれらを処理する専用ハードウェア[6)~8)]の開発等に分類できる．

さらにPROLOGプログラムの並列処理方式は，引数間並列[7)]，AND並列[8)]，OR並列[1)~3),6)]の3つに大別される．引数間並列では，並列に処理する処理単位のレベルが非常に小さく，無矛盾性チェックが必要とされるため処理に伴うオーバヘッドが大きいことが予想される．またAND並列でも，共有変数が存在する場合の無矛盾性チェック，あるいはストリーム並列に伴う同期・データ転送によるオーバヘッドが大きいと考えられる．したがってこれらの手法を実現するシステムでは高速なデータ転送を可能とする特殊なアーキテクチャの開発が望まれる[7),8)]．

OR並列は，前者2つの並列処理方式と比べると処理単位（グラニュラリティ）を大きくとれ，しかも各処理単位が独立であるということから，特殊なアーキテクチャを持たないマルチプロセッサシステム上でも実現可能である．OR並列は，一般的に幅優先あるいは深さ優先の2種の探索方法を用いて実現される．幅優先探索の問題点としては，まず第1に，適切な解を1つ見つける場合に複数のプロセッサを用いているのに1台のプロセッサによる逐次処理に要する時間よりも長時間を要するという減速異常[9)]を起こす可能性があることである．第2に並列性の爆発を起こす可能性があるということである．第3には，例えば探索の深さと共にゴールの書換えを行っていくような場合，ゴールの大きさが大きくなり，データ転送に伴うオーバヘッドが大きくなるということである．これに対して，深さ優先探索は幅優先探索に比べさらに処理単位を大きくすることができるとともに，幅優先探索とは逆に解を1つ見つければ良い場合にm台のプロセッサを用いて1台の時の$1/m$以下の処理時間を得る加速異常[9)]を引き出すことができる．しかし，この方法でも，負荷のプロセッサへの割当て（スケジューリング）に伴い履歴のコピー等に要するオーバヘッドが問題となる．

以上述べたように，並列処理を実現する場合には，プロセッサ間のデータ転送オーバヘッドあるいは負荷分散（スケジューリング）のオーバヘッドを最小化する並列化手法の開発が重要である．また従来のPROLOG処理系を見た場合，専用アーキテクチャの開発が多く行われてきた．しかし，システムとしてのコストパフォーマンスを考えるとき，上に挙げたオーバヘッドを低く抑える並列処理方式が存在するならば，汎用的なプロセッサを複数結合したマルチプロセッサ方式の並列処理システムの開発も必要と考える．以上のことを考慮し，本論文では，マルチプロセッサシステム上で，スケジューリング，プロセッサ間通信

† An OR Parallel Processing Scheme of PROLOG Using Hierarchical Pincers Attack Search by MUNENORI KAI, KAZUO KOBAYASHI and HIRONORI KASAHARA (Department of Electrical Engineering, School of Science and Engineering, Waseda University).

†† 早稲田大学理工学部電気工学科
* 現在 成蹊大学工学部経営工学科
Now with Department of Industrial Engineering, Faculty of Engineering, Seikei University.

などによるオーバヘッドを低く抑えたPROLOG の効率良い並列処理を可能とする手法として，「階層型挟み打ち探索法」を提案する．まず2章で，本論文でモデルとするマルチプロセッサシステムについて述べ，3章で「階層型挟み打ち探索法」を詳述する．4章では具体的なプログラム例を用いてシミュレーション評価を行った結果を示す．最後に5章で本手法の有効性と今後の課題についてまとめる．

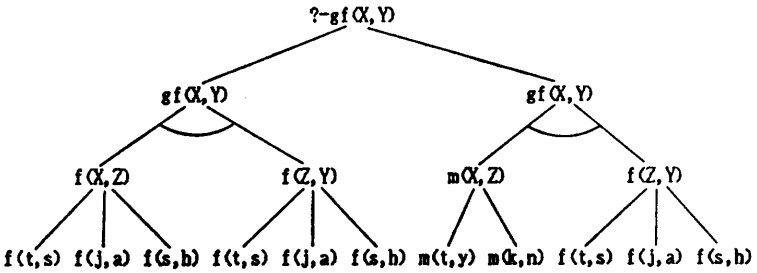

図3 図2で ?-gf(X,Y). に対する AND/OR 木
Fig. 3 The AND/OR-tree of Fig. 2 with the question "?-gf (X, Y).".

2. 並列処理システムのモデル

ここでは密結合マルチプロセッサシステムのモデルとして，図1のように数十台程度までの比較的少数のプロセッサエレメント PE，共有メモリ CM およびコントロールプロセッサ CP が1本あるいは複数のバス等で結合されたシステムを考える．各 PE および CP は各自のプログラムおよびデータを記憶するローカルメモリと，プロセッサ間での1対1データ転送およびデータのブロードキャストを実現するための2ポートメモリを持つものとする．この2ポートメモリは，本論文で提案する並列処理手法の実現において，読み書きを同時に行えるので，プロセッサ間のデータ転送に伴うオーバヘッドを下げることができる．また，各 PE のローカルメモリは，単一の PE で PROLOG の逐次処理を行うのに十分な容量を持つものとする．CP は実行時にプロセッサへの負荷のダイナミックな割当て（スケジューリング）を行う．

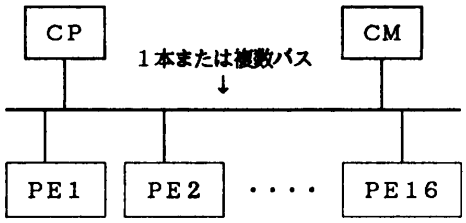

図1 汎用目的マルチプロセッサシステム
Fig. 1 An example of multiprocessor system.

```
father(taro, satoshi).                    略記
father(jiro, akio).                       f(t,s).
father(satoshi, hiroshi).                 f(j,a).
mother(taro, yoshiko).                    f(s,h).
mother(keiko, noriko).                    m(t,y).
grandfather(X,Y):-                        m(k,n).
      father(X,Z), father(Z,Y).           gf(X,Y):-
grandfather(X,Y):-                          f(X,Z), f(Z,Y).
      mother(X,Z), father(Z,Y).           gf(X,Y):-
                                            m(X,Z), f(Z,Y).
```

図2 PROLOG プログラムの例
Fig. 2 An example of PROLOG program.

3. 階層型挟み打ち探索法

3.1 OR 木による探索空間の表現

PROLOG の処理過程は，一般に AND/OR 木で表される．これを，AND 逐次・OR 並列を表現した OR 木に変換する．図2の簡単なプログラムでその例を示す．このプログラムは，いくつかの親子関係を表す事実と，祖父・孫関係を表す規則とからなっている．表記の簡略化のため，各節を右のような省略形で表すことにする．このようなプログラムに対して，?-grandfather (X, Y). という質問を行った場合の探索過程は図3の AND/OR 木となる．この AND/OR木で，AND 関係のノードは逐次に，OR 関係のノードは並列に実行するならば，AND の枝を縦方向に展開することによって，図4のような OR 木に変換することができる．各ノードの上部は単一化を試みられる節を表し，下部は次に単一化を行うゴールを表す．したがって初期ゴールを表す根ノードでは上部が空であり葉ノードでは下部が空となる．この OR 木では，根ノードからどの葉ノードへの経路（パス）もすべて並列に独立に探索可能である．

3.2 階層型挟み打ち探索法の原理

図5に階層型挟み打ち探索法における OR 木のプロセッサへの割当て方式を示す．

本手法ではまず，n 台（図1では 16 台）の PE を2つのグループ，グループ1とグループ2に分ける．そしてこの2つのグループが OR 木の左右から挟み打ちの形で探索を行っていく．このとき，グループの PE 台数は常に等しく分ける必要はなく，OR 木の左

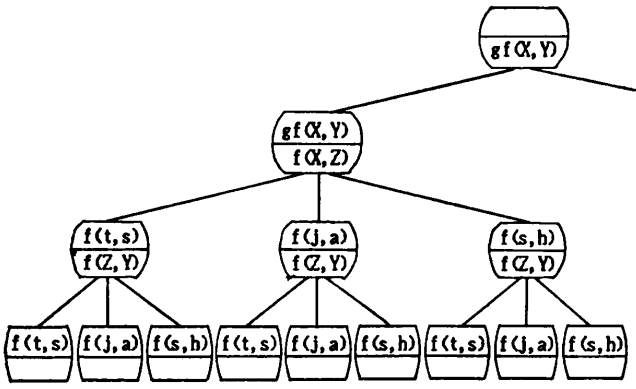

図 4 図3に対する OR 木
Fig. 4 OR-tree for Fig. 3.

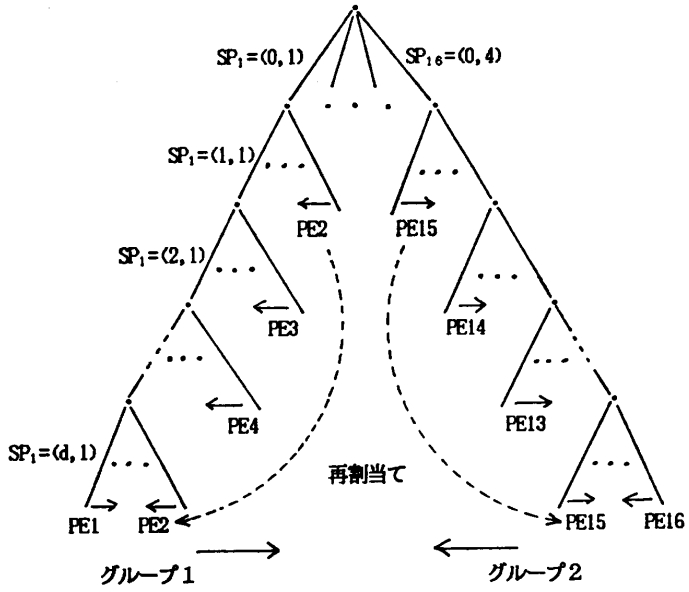

図 5 OR 木のプロセッサへの割当て方式
Fig. 5 Hierarchical pincers attack search for OR-tree.

右どちら側からの探索に重点を置くかによって変えることができる．以下では，図1の16台の PE を8台ずつのグループに分けた場合で述べる．

次に各グループ内での PE の探索形態について述べる．まずグループ1では，PE 1 をリーダ PE とし，PE 2～PE 8 を PE 1 のスレーブ PE とする．同様にグループ2では，PE 16 をリーダ PE とし，PE 9～PE 15 をスレーブ PE とする．そしてすべての PE は同じ定義節を持つ．グループ1において，リーダ PE である PE 1 は OR 木の左から右へ逐次処理と同様の深さ優先探索を行う．PE 2～PE 8 のスレーブ PE は PE 1 の探索パス上の各ノードを根ノードとする部分探索木を右から左へ，PE 1 と挟み打ちをする形で，深さ優先探索を行う．ある部分探索木の探索が終了したかどうかのチェックは，リーダ PE の探索領域とスレーブ PE の探索領域が重複したかどうか調べることによって行われる．このようにスレーブ PE を部分探索木の右側に割り当てる操作と，探索領域の重複のチェックを各 PE で独立に行えるようにするため，次に示すセレクションポインタ (SP) というもので各 PE の探索位置を管理する．

SP_1=(OR 木の深さ，OR 枝番号)

これは，リーダ PE である PE 1 が生成する SP を示している．スレーブ PE も同様の SP を生成するが，両グループのリーダ PE のみが OR 木の探索を1つ進めるごとにこの SP を他の PE と CP にブロードキャストする．ここで，OR 木の深さとは，OR 木の根ノードを深さ0とし，以後，枝を1つ進めるごとに1ずつ増えるというものである．また，OR 枝番号とは，ある深さのノードにおいて次に左から何番目の OR 選択枝を選ぶかということを表している．例えば，図5で，PE 1 は深さ0で左から1番目の枝を選択したので SP_1=(0, 1) をまずブロードキャストし，次に深さ1で左から1番目の枝を選択するので SP_1=(1, 1) をブロードキャストすることを示している．ブロードキャストされた SP は各スレーブ PE および CP 上で**図 6** に示すようなテーブルの形式 (SP テーブル) で保持される．この SP テーブルは，リーダ PE のユニファイ成功により増長し，バックトラックにより

深さ	OR枝番号
0	1
1	1
2	1
3	2
4	1
⋮	⋮

図 6 CP および PE 内の SP テーブル
Fig. 6 SP tables in CP and PE's.

縮退する．スレーブ PE は，CP から未探索領域を割り当てられるまで，リーダ PE から送られた SP テーブルに従ってリーダ PE と同じパスを進んでいく．そして CP からリーダ PE の探索パス上の1つのノードの深さを指定されると，そのスレーブ PE は指定された深さでそこのノードを根ノードとする部分探索木をリーダ PE と反対側の枝からリーダ PE と挟み打ちを行う形で探索を進めていく．例えば図5で，PE2 が深さ1で探索領域を割り当てられ，OR 選択枝が5本あったとすると，PE2 は $SP_2=(1, 5)$ というポインタを自身内部のみで生成しながら，右から左に向けて深さ優先探索を行っていく．そして順に $SP_2=(1, 4), \cdots, (1, 3), \cdots, (1, 2), \cdots, (1, 1)$ と生成した時点で，リーダ PE の SP と自分の SP が一致したことがわかる．PE2 はここでその部分探索木の探索がリーダ PE の進んだパスを除いてすべて終了したことを CP に報告し，割当て待ち状態に入る．割当て待ち状態に入った PE は休止するわけではなく，リーダ PE からすでに送られていた SP に従ってリーダ PE の探索パスを追従し，次に CP から割り当てられる探索領域の探索に必要な履歴（環境）を自己再生する．

また，リーダ PE がすでにあるスレーブ PE を割り当てたノードにバックトラックし，そのスレーブ PE の探索中の部分木に入った場合を**図7**に示す．この場合，スレーブ PE はリーダ PE から送られた SP により自分のパスとリーダ PE のパスが一致している部分がわかるので，その部分に含まれるノードごとに自分の割当て深さを1ずつ増し，CP にそれらの深さでの探索が終了していることを知らせる．これを受けた CP はそれらのノードを，他の割当て待ち状態のスレーブ PE に割り当てないようにする．このスレーブ PE による操作を割当て深さの更新と呼ぶ．

グループ2では，リーダ PE とスレーブ PE の挟み打ちを行う方向が，グループ1とは左右逆になる．グループ1とグループ2の探索領域が重複した場合には，後から相手の探索領域に到達したグループが重複を検出し，もう一方の探索中のグループの1つのスレーブとして探索を行う．例えばグループ1が重複を検出した場合，グループ1がグループ2のリーダ PE のパスを追従し，CP からグループ2のリーダ PE の探索パス上の深さを受け取ると，その深さのノードを根ノードとする部分探索木でグループ1が階層型挟み打ち探索を行うことになる．

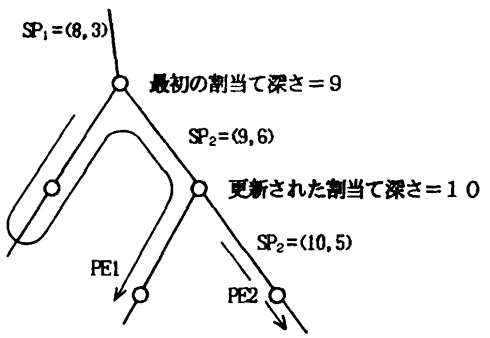

図7 スレーブ PE の割当て深さの更新
Fig. 7 Update of assigned depth of slave PE.

全体の探索の停止は，解の求め方の要求によって異なる．初期ゴールを満足する解を1つ見つければよい場合には，どれかのプロセッサが解を見つけた時点ですべての探索は停止する．また，全解探索を行う場合には，あるプロセッサが解を見つけたとしても，すべての探索パスを調べ終るまでは階層型挟み打ち探索を続ける．

本手法では，次探索領域の割当て（スケジューリング）は，CP がスレーブ PE に対してリーダ PE の探索パス上の深さのみを指定すればよく，履歴のコピーなどは必要ない．さらに，1回に割り当てる探索領域すなわちタスクグラニュラリティは非常に大きく，ダイナミックにタスクをスケジュールする頻度を少なくすることができるため，スケジューリングによるオーバヘッドも低く抑えることができる．

また，図5の OR 木において深さが1つ進むということは，新しいゴールを1つ生成するということに相当し，左右から挟み打ちを行うという動作はプログラムでみれば同一ヘッドを持つ節の集合を上から順に選択していくことと，下から順に選択していくことに相当する．したがって，各プロセッサに要求される記憶領域も各プロセッサが逐次処理を行った場合とほぼ同じである．

3.3 カットと再帰の扱い

PROLOG のプログラムでは，実際，カットや再帰のようにその節の選択順序が意味を持つ場合が多い．本節では，このように選択順序に制約がある場合をどのように階層型挟み打ち探索法で扱うかということについて述べる．

図8にカットを含んだプログラムの1例とその OR 木を示す．ただし，図8では，簡略化のため節は述語部のみで表し，引数部は省略した．

逐次処理の場合のカットの働きは，カットを通過したときに現在注目している節のカットまでの探索履歴

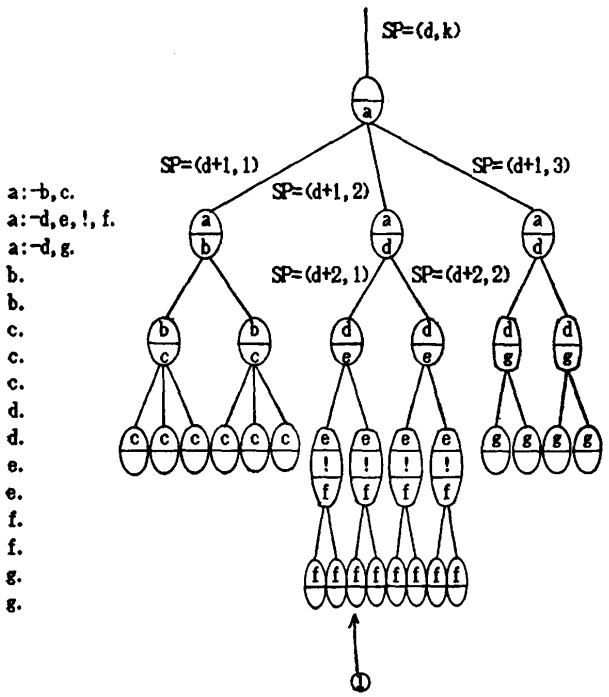

図 8 カットを含むプログラムとその OR 木
Fig. 8 Program and OR-tree with cut.

を最終決定とみなすことである．その結果，例えば図8の OR 木で①のパスに探索が進められてそのカットを通過したとき，そのカット以下の部分木を除いて右側にあるすべての部分木は探索の対象とはならない．実際のプログラムでは，このような OR 木が部分探索木としてより大きな OR 木に含まれるわけであるが，カットの働きを損なうことのないように，次のようにして階層型挟み打ち探索を行っている．

すなわち，カットをボディ部に含む節と同一のヘッドを持つ節の集合を OR 選択枝として持つノードにおいては，リーダ PE・スレーブ PE を問わず常に次の枝の選択を左から行う．そして深さが1つ深くなった次のノードから階層型挟み打ち探索を行う．図8を用いた具体的な動作を以下に示す．

（1） リーダ PE が SP=(d, k) を生成し，図8の根ノードに達した場合

この部分探索木の根ノード a では，CP はスレーブ PE を右側の探索領域に割り当てない．リーダ PE 自身は SP=(d+1, 1) を生成かつブロードキャストし，次のノードに進む．次のノードを根ノードとする部分探索木については，通常の階層型挟み打ち探索を行う．

SP=(d+1, 1) の枝の先の部分探索木の探索終了後，リーダ PE がバックトラックして SP=(d+1, 2) の枝を進み，a/d のノードについて階層型挟み打ち探索を行うが，このようにすべてのパスがカットを含む部分木ではグループによってその動作が異なる．

(1.1) グループ1の場合

リーダ PE は SP=(d+2, 1) を生成し，左から右へ深さ優先探索を行う．また，割当て待ち状態のスレーブ PE があれば SP=(d+2, 2) の枝を割り当てられ，右から左へ深さ優先探索を行う．左から右へ進むリーダ PE がカットを含むノードを通過した場合，ノード a/d 以後割り当てられたすべてのスレーブ PE の探索を中止させる．探索を中止したスレーブ PE は割当て待ち状態となり，リーダ PE のパスを追従する．また，もしリーダ PE がカットを通過する前にスレーブ PE がカットを通過した場合，逐次処理によって得られる解と解の一致性を保つため，そのパスの成功・失敗にかかわらず左方向に深さ優先探索を続ける．このとき，途中で解が見つかっても左側にあるカットの影響により解となるのか判断がつかないので，その解をローカルメモリ上に一時保管する．ただし，バックトラックによりスレーブ PE 自身が現在通過してきているカットより左側のカット以下の部分木の探索を開始した場合には，それまでに見つけた解を放棄し，新しく通過したカット以下の部分木で得られる解を一時保管する．そして，もしそのパスより左側のカットがすべて通過前に切り捨てられたならば，一時保管された解を真の解とする．

このようにしてノード a/d を根とする部分探索木が階層型挟み打ち探索を行われるが，もしどのカットも通過することなくすべてのパスが失敗に終れば，リーダ PE は SP=(d+1, 3) の枝から先で階層型挟み打ち探索を行う．

(1.2) グループ2の場合

リーダ PE とスレーブ PE の左右の探索方向が挟み打ちを行うときに入れ換わるだけでグループ1と同じである．ただし，図8の SP=(d+1, 2) の選択による部分探索木を探索する場合はグループ1とは異なる．リーダ PE は SP=(d+2, 1) を CP を解して割当て待ち状態のスレーブ PE に割り当てる．右から左に探索を進めるリーダ PE のカット通過による解の報告の仕方は，グループ1のスレーブ PE の場合と同様である．た

だし，リーダ PE が左に割り当てたスレーブ PE がカットを通過した場合，そのスレーブ PE はリーダ PE を含めて自分より右にあるすべての PE に CP を介して探索打ち切りを指示する．打ち切り指示を受けるとスレーブ PE は割当て待ち状態となり，リーダ PE は指示を出したスレーブ PE を割り当てたノードまでバックトラックする．そして図7で示したようにスレーブ PE の割当て深さの更新が行われ，以降の部分探索木で挟み打ち探索を行う．

（2） スレーブ PE が図8の根ノードに到達した場合

この場合には，スレーブ PE はリーダ PE が後からこの部分探索木に到達するまで単独で深さ優先探索を行う．そのスレーブ PE がグループ1に属し，右から左へ探索を進める場合，解の一時保管の仕方は(1)と同様である．またスレーブ PE がグループ2に属する場合には通常の逐次処理と同じ深さ優先探索を行う．リーダ PE が後からこの部分木に到達した場合は，(1)と同様に探索を行う．

再帰を含むプログラムでは，一般に，再帰呼び出しされる節(これを再帰節と呼ぶ)よりも，境界条件を示す節の方が先に単一化を試みられる．**図9**は再帰節を含む簡単なプログラムとその OR 木を示している．図中2重線で囲まれたノードは再帰節と同じヘッド部の述語を持つ節集合と単一化可能なゴールが生じることを表す（これを再帰ノードと呼ぶことにする）．すなわち，図9の部分木では葉ノードとなっている再帰ノードは，その部分探索木と同じ部分木を繰り返すということを表している．

この部分探索木では，右から左への深さ優先探索は無限ループに陥る可能性を含んでいる．したがって，

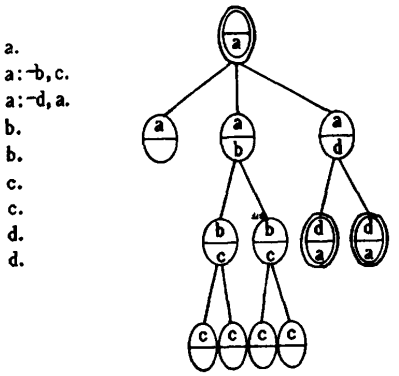

図 9 再帰を含むプログラムとその OR 木
Fig. 9 Program and OR-tree with recursion.

再帰ノードを根ノードとする部分探索木では次のような方法で階層型挟み打ち探索を行う．

再帰ノードに達した PE は，次の枝の選択を行うときに限り，その PE がグループ1，2のどちらに属するか，あるいはリーダ，スレーブのどちらであるかにかかわらず，必ず最左端の枝から選択するものとする．したがってリーダ PE の場合，再帰ノードではスレーブ PE の割当ては行わない．そして，再帰ノードから1つ深さの深くなったノードからは通常の階層型挟み打ち探索を行う．

4. 階層型挟み打ち探索法の評価

本章では，3章で述べた階層型挟み打ち探索法のシミュレーションを行い，具体的なプログラムに適用してその並列処理性能を調べた結果について報告する．

4.1 対象プログラム

ここで取り上げたプログラムは数式を微分するプログラムである．すなわち，

?-d(log(3*x^2−cos(5*x)), x, F).

[第1引数の式をxで微分するとFになる]

という質問に対して，

F=(6*x+5*sin(5*x))/(3*x^2−cos(5*x))

のように微分の結果の式を与えるものである．このプログラムは，整式や三角関数・指数関数・対数関数の微分公式を記述した 16 個の規則と，係数をまとめて最簡形にするための151個の規則および事実からなる．また微分公式の 16 規則中 15 個はヘッド部の述語が等しい OR 関係にあり，最簡形のための151個の規則・事実中145個が OR 関係にある．規則の最大ボディリテラル数は4でカットや再帰を含むものも存在する．

このプログラムに対する OR 木の形状は，節の並ぶ順のため，探索領域が横方向に非常に大きく，深さが OR 木の左側の方が深いという特徴を持っている．このような OR 木に対して階層型挟み打ち探索のシミュレーションを行った．また，このシミュレーションにおける1単位時間を，OR 木のノードを1つ進める時間，すなわち，あるゴールに対して OR 関係の節への SP を生成し，単一化を試みる時間としている．ここでは，CP がスレーブ PE に部分探索木を割り当てるための時間およびそれに伴うデータ転送時間は各スレーブ PE が次探索領域の探索のために必要な履歴を自己生成する時間より小さいと考えて，そのオーバヘッドを評価しているが，リーダ PE が単一化の度に

ブロードキャストする SP の転送時間は無視できるものとしている．評価指標としては，最適解すなわち最簡形の式を得るまでの並列処理時間と，全解探索（この問題では唯一に決まる最適解を与えるパスが複数存在する場合があるため）を行うのに要する並列処理時間を取り上げた．

4.2 評価結果

シミュレーションの結果を図 10 に示す．図 10 で横軸はグループ1とグループ2の PE 台数の合計を表している（グループ1，グループ2の PE 台数をそれぞれ m1, m2 とする）．図中 m1≠m2 とは PE 台数が8台以下ではそれらをすべてグループ1の PE とし，9台以上からグループ2の PE を1台ずつ増やした場合を表している．m1=m2 は，両グループの PE 数が等しい場合である．縦軸は，1PE による処理時間を基準としたときの階層型挟み打ち探索による並列処理時間の割合を百分率で示してある．したがって，50% ならば 1PE による処理時間の半分の並列処理時間が得られたことを意味する．

このグラフから全解探索の場合，PE が5台程度までは台数に比例した並列処理の速度向上が得られた．PE 16 台では約8倍の速度向上しか得られていないが，これはこのプログラムの OR 木の深さがあまり深くないために，PE 台数分の並列性を十分に引き出せなかったためと考えられる．また最適解を得るにあたっては，PE 2 台で 50 倍，PE 16 台では約 111 倍という PE 台数以上の速度向上すなわち加速異常が得られた．これは 1PE による左側からの深さ優先探索に比べて，特に右からの深さ優先探索により解が得られる場合に生じる現象である．このことは解の位置が OR 木の左右どちらに寄っているかによって得られる速度向上が異なることを示している．また，本手法は，深さ優先探索に基づく OR 並列処理を行っているので，幅優先探索の場合と違って，本質的に探索方法による減速異常は起こさない．すなわち，幅優先探索では複数のプロセッサを用いても，単一プロセッサの場合より処理が遅くなることがある[1]が，本手法では，達成される並列処理時間は，最悪でも単一プロセッサによる深さ優先探索で得られる処理時間より長くはならないことが保証されている．

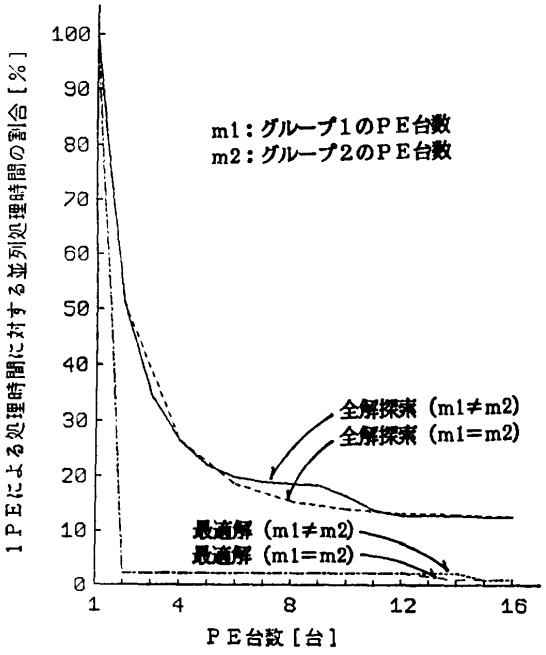

図 10 PE 台数による並列処理時間の減少度
Fig. 10 Number of PE vs parallel processing time.

次にスケジューリングによるオーバヘッドであるが，本手法では常に部分探索木を左方向と右方向に2分するため，タスクグラニュラリティを大きくとれ，ダイナミックスケジューリングを行う頻度を少なく抑えることができる．例えば，図 10 で示した数式微分プログラムの並列処理においては，PE 4台ずつの2つのグループ（計8台）で階層型挟み打ち探索を行った場合，平均して約 1400 ユニフィケーションに1回，PE 8台ずつ（計 16 台）の場合は，平均して約 600 ユニフィケーションに1回しかスケジューリングが行われないという結果が得られた．これに対し，同じ深さ優先探索に基づく他の OR 並列処理手法である株分け並列推論方式[3]では，スケジューリング頻度が非常に高く，数十ユニフィケーションに1回以上スケジューリング（株分け処理）が生じることが 30〜40% の割合であることが報告されている．このことから階層型挟み打ち探索法は，スケジューリング頻度を極めて低く抑えていることがわかる．次に，1回のスケジューリングに要求されるオーバヘッドについて考えてみる．幅優先探索に基づく Paralog[1] やゴール書換えモデル[2]，あるいは株分け並列推論方式の場合には，ゴールや探索領域をプロセッサに割り当てるときに過去の履歴も一緒に渡す必要があり，この転送によるオーバヘッドが大きくなってしまう．また，深さ優先で，左の方のパスから順にプロセッサを割り当てるという単純な方法を考えた場合でも，探索の終了したプロセッサを再割当てするために，他のすべてのプロセッサの探索した部分をトレースしておく必要があ

り，そのためのプロセッサ間のデータ転送回数や割当て位置を決定するオーバヘッドおよび履歴のコピーに要するオーバヘッドが大きくなってしまうことになる．これに対して，階層型挟み打ち探索法では，1回のスケジューリングで要求されるデータ転送は，CPがスレーブ PE に対してリーダ PE の探索パス上の深さのみを指定する1データのみである．また，セレクションポインタ SP のブロードキャストも，1ユニフィケーションにつきわずか2データ（OR 木の深さと OR 枝番号）の転送にすぎないので，このオーバヘッドも十分小さいものと考える．そして，割り当てられたスレーブ PE は，この SP を用いて必要な履歴を自己生成しているので，リーダ PE から履歴をコピーする必要がなく，一度割り当てられると，探索を開始して終了し，次の再割当て時のための履歴の生成まで，一切他の探索中のプロセッサをわずらわせることなく独立に行うので，他の手法に比較してオーバヘッドを大幅に低減できることがわかる．

次に左右から挟み打ちを行う2つのグループの PE 台数の配分の仕方による並列処理効果への影響について述べる．全解探索を行う場合に，OR 木が左右どちらに深いか，その深い方から探索を始めるグループに PE を多く配分した方がよいと考えられるが，実行時に OR 木が本当に深くなるかどうかが明らかではないため，一般には2グループの PE 台数を等しくするのが良いと考える．

5. おわりに

本論文では，PROLOG の OR 並列処理手法として階層型挟み打ち探索法を提案した．この手法は以下のような特長を持つ．

①各プロセッサは本質的に深さ優先探索を行うので，従来の逐次処理のための高速化手法が利用できる．

②各プロセッサ上で独立に処理できる並列処理単位（タスクグラニュラリティ）を大きくとることができる（部分探索木レベル）．このため，実行時に部分探索木をプロセッサへ割り当てるダイナミックスケジューリングの頻度すなわちスケジューリングに伴うオーバヘッドを低く抑えることができる．

③SP を用いて，部分探索木のプロセッサへの割当て制御を行うため，1回のスケジューリングに要するオーバヘッドをさらに小さく抑えることができる．

④プロセッサは割当て待ち状態のときに次の探索に必要な履歴（環境）を，SP を用いて自己再生する．このため，割当て時にプロセッサ間で履歴をコピーする必要がなく，データ転送によるオーバヘッドを低減することができる．

⑤以上のような特長を持つことから，専用のハードウェアを必要とせず，汎用のマルチプロセッサシステム上でも，種々のオーバヘッドの少ない効率良い並列処理を実現することができる．

今後，実マルチプロセッサシステム上で本手法を実現することにより，データ転送およびスケジューリングに伴うオーバヘッドをより詳細に考慮した評価を行い，本手法の有効性および実用性をチェックする計画である．

謝辞 本研究遂行にあたり多くの後助力をいただいた早稲田大学理工学部電気工学科成田誠之助教授に感謝いたします．また本学情報科学研究教育センター，AI 研究会において日頃有益な御助言をいただく数学科廣瀬健教授，電子通信学科小原啓義教授，村岡洋一教授，電気工学科白井克彦教授に感謝いたします．

参考文献

1) 相田 仁ほか：並列 Prolog 処理システム "Paralog" について，情報処理，Vol. 24, No. 6, pp. 830-837 (1983).
2) 後藤厚宏ほか：ゴール書換えモデルに基づく論理型プログラムの並列処理方式，情報処理，Vol. 25, No. 3, pp. 413-419 (1984).
3) 増沢秀穂ほか：株分け並列推論方式とその評価，*Logic Programming Conf. '86*, ICOT, pp. 193-200 (1986).
4) Shapiro, E.: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report, TR-003 (1983).
5) Ueda, K.: Guarded Horn Clauses, *Logic Programming Conf. '85*, ICOT, pp. 225-236 (1985).
6) Motooka, T. et al.: The Architecture of a Parallel Inference Engine—PIE—, *Proc. Int. Conf. FGCS*, ICOT, pp. 479-488 (1984).
7) Ito, N. and Shimizu, H.: Data-flow Based Execution Mechanisms of Parallel and Concurrent Prolog, *New Generation Computing*, Vol. 3, No. 1, pp. 15-41 (1985).
8) 後藤厚宏：並列推論マシン PIM，電気・情報関連学会連合大会，5, pp. 49-52 (1987).
9) Wah, B. W. et al.: Multiprocessing of Combinatorial Search Problem, *IEEE Comput.*, Vol. 18, No. 6, pp. 93-108 (1985).

(昭和62年10月14日受付)
(昭和63年5月10日採録)

甲斐 宗徳（正会員）

昭和35年生．昭和58年早稲田大学理工学部電気工学科卒業．昭和63年同大学院理工学研究科電気工学専攻博士課程修了．工学博士．昭和62年早稲田大学理工学部電気工学科助手．昭和63年成蹊大学工学部経営工学科助手，現在に至る．マルチプロセッサシステムにおけるダイナミックスケジューリングアルゴリズム，知識情報処理，論理プログラミング言語の並列処理の研究に従事．電気学会，電子情報通信学会各会員．

小林 和男（学生会員）

昭和39年生．昭和62年早稲田大学理工学部電気工学科卒業．現在，同大学院修士課程に在学中．Prologの並列処理に関する研究に従事．

笠原 博徳（正会員）

昭和32年生．昭和55年早稲田大学理工学部電気工学科卒業．昭和60年同大学院博士課程修了．昭和58～60年同大学電気工学科助手．昭和60年日本学術振興会特別研究員．昭和61年早稲田大学理工学部電気工学科専任講師，昭和63年助教授，現在に至る．1987 IFAC WORLD CONGRESS 第1回 Young Author Prize 受賞．マルチプロセッサ・スケジューリング・アルゴリズム，並列処理，ロボット制御，シミュレーション等の研究に従事．電子情報通信学会，電気学会，ロボット学会，シミュレーション学会，IEEE，ACM 各会員．工学博士．