マイクロコンピュータ 48－1
計算機アーキテクチャ 69－1
（1988. 1. 21）

Prolog OR並列処理手法 ― 階層型挟み打ち探索法 ―

甲斐 宗徳,　小林 和男,　笠原 博徳

早稲田大学理工学部電気工学科

本稿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なう。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な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なうため、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。本手法の性能及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。

AN OR PARALLEL PROCESSING SCHEME OF PROLOG - HIERARCHICAL PINCERS ATTACK SEARCH -

Munenori KAI, Kazuo KOBAYASHI, Hirinori KASAHARA

Department of Electrical Engineering,
School of Science and Engineering, Waseda University

3-4-1, Ohkubo, Shinjuku, Tokyo, 160 Japan

We proposes an OR parallel processing scheme of Prolog named "Hierarchical Pincers Attack Search". In the scheme, an OR-tree, which represents an execution process of a Prolog program, is searched from right and left by a plurality of processors.

Each processor does the depth first search independently. The pincers attack search allows us to get a coarse task granularity. That reduces the frequency of the task assignment or the task scheduling, and also the amount of the data transfers among the processors. Furthermore, the introduction of a special pointer which indicates the status of the processors, minimizes the data transfers caused by the task scheduling.

In addition, the depth first searches from the both sides extract the acceleration anormaly efficiently. The effectiveness of the proposed scheme is confirmed by simulations of the parallel processing process.

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なうためには、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なわれている。これらの研究は、PROLOGの並列処理手法[1),2),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),7),8)]の開発等に分類できる。

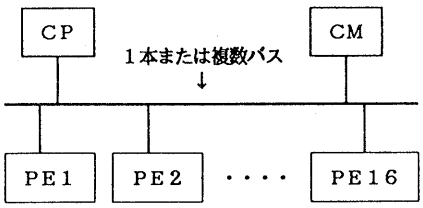
さらにPROLOGプログラムの並列処理方式は、引数間並列[7)]，ＡＮＤ並列[8)]，ＯＲ並列[1),2),3),6)]の３つに大別される。引数間並列では、並列に処理する処理単位のレベルが非常に小さく、無矛盾性チェックが必要とされるため処理に伴うオーバヘッドが大きいことが予想される。またＡＮＤ並列でも、共有変数が存在する場合の無矛盾性チェック、あるいはストリーム並列に伴う同期・データ転送によるオーバヘッドが大きいと考えられる。従ってこれらの手法を実現するシステムでは高速なデータ転送を可能とする特殊なアーキテクチャの開発が望まれる[7),8)]。

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、汎用の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な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なわれてきたが、システムとしてのコストパフォーマンスを考えるとき FORTRANのような科学技術計算と論理型言語の両方の並列処理を効率良く行なえるシステムの開発も必要と考える。以上のことを考慮し、本稿では、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。また、本手法を実現可能な汎用的なマルチプロセッサシステムのモデルについても述べる。さらに、具体的なプログラム例を用いて「階層型挟み打ち探索法」のシミュレーション評価を行なった結果を報告し、本手法の有効性を示す。

2．並列処理システムのモデル

ここでは汎用型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及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及びＣＰは各自のプログラム及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データ転送及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。このとき、各ＰＥのローカルメモリは、単一のＰＥでPROLOGの逐次処理を行なうのに十分な容量を持つものとする。また、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）を行なう。



CP：Control Processor
CM：Common Memory
PEn：Processor Element

図1　汎用目的マルチプロセッサシステム

3．階層型挟み打ち探索手法

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なうゴールを表す。従って初期ゴー

```
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

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）も全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なっていく。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、ＰＥ１をリーダＰＥとし、ＰＥ２～ＰＥ８をＰＥ１のスレーブＰＥとする。同様にグループ２では、ＰＥ１６をリーダＰＥとし、ＰＥ９～ＰＥ１５をスレーブＰＥとする。そして全ての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ＰＥ１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なう。ＰＥ２～ＰＥ８のスレーブＰＥはＰＥ１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、ＰＥ１と挟み打ちをする形で、深さ優先探索を行なう。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なわれる。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で独立に行な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_i ＝（ＯＲ木の深さ，ＯＲ枝番号）

これは、リーダＰＥであるＰＥ１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つ進める毎にこの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つ進める毎に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、ＰＥ１は深さ０で左から１番目の枝を選択したのでSP_i ＝ (0, 1) をまずブロードキャストし、次に深さ１で

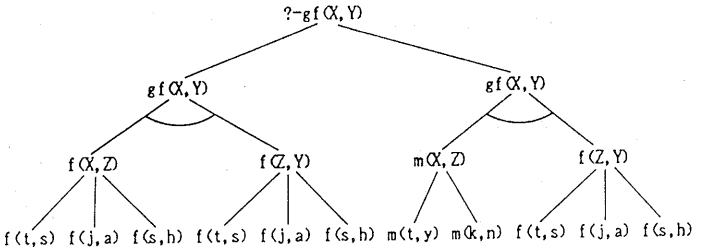


図３　図２で?-gf(X,Y). に対するＡＮＤ／ＯＲ木

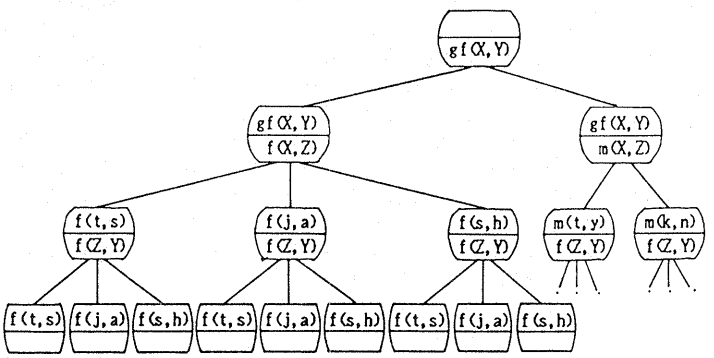


図４　図３に対するＯＲ木

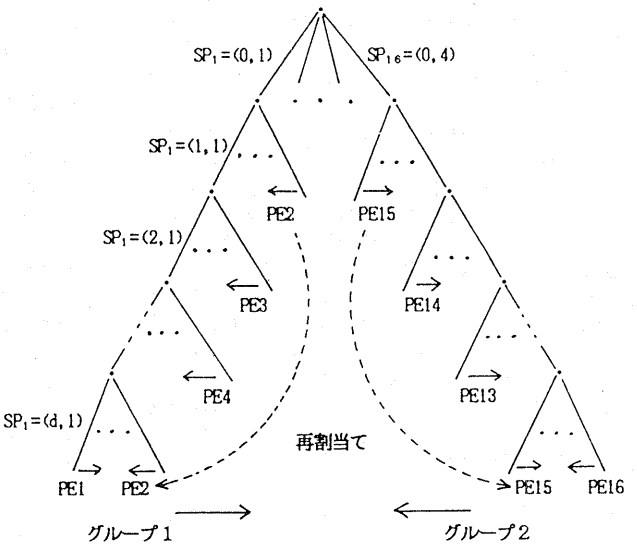


図５　ＯＲ木のプロセッサへの割当て方式

深さ	OR枝番号
0	1
1	1
2	1
3	2
4	1
⋮	⋮

図6　CP及びPE内のSPテーブル

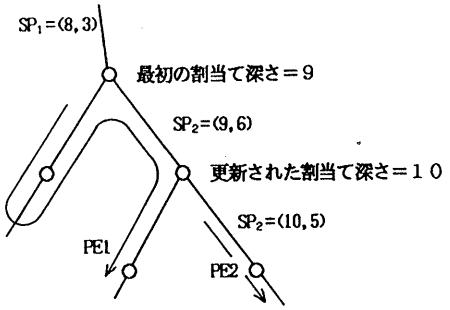


図7　スレーブPEの割当て深さの更新

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及び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。スレーブPEは、CPから未探索領域を割当てられるまで、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と挟み打ちを行な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なっていく。そして順にSP_2 = (1, 4)，…，(1, 3)，…，(1, 2)，…，(1, 1) と生成した時点で、リーダPEのSPと自分のSPが重複したことが分かる。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全て終了したことを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毎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は、リーダPEとスレーブPEの挟み打ちを行なう方向が、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なう。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なうことになる。

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つ見つければ良い場合には、どれかのプロセッサが解を見つけた時点で全ての探索は停止する。また、全解探索を行なう場合には、あるプロセッサが解を見つけたとしても、全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良く、履歴のコピーなどは必要ない。さらに、1回に割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な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。従って、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な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、これより右にある全ての部分木はたとえ①のパスが失敗に終っても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なっている。

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なう。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なう。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なう。

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なうが、このように全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なう。また、割当て待ち状態のスレーブＰＥがあればＳＰ＝（d+2, 2）の枝を割当てられ、右から左へ深さ優先探索を行なう。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以後割当てられた全てのスレーブ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、ローカルメモリ上に一時保管する。そして、もしそのパスより左側のカットが全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。またスレーブＰＥにより複数の解が発見される場合には最新の解のみを一時保管し、前の解は放棄する。

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全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なう。

(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な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全ての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なわれ、以降の部分探索木で挟み打ち探索を行なう。

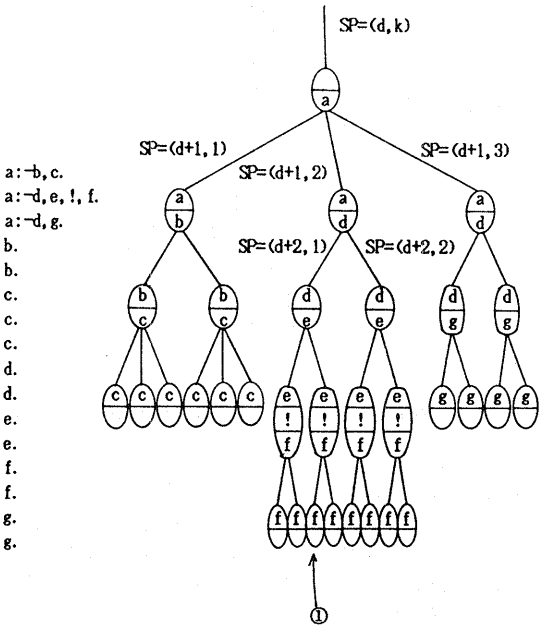


図８　カットを含むプログラムとそのＯＲ木

(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なう。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なう。リーダＰＥが後からこの部分木に到達した場合は、(1) と同様に探索を行なう。

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。従って、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

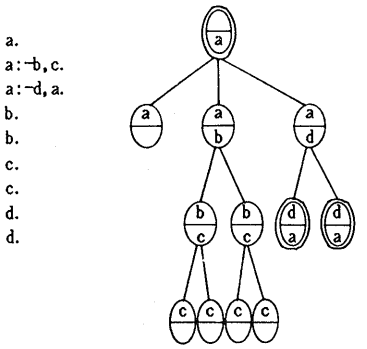


図9　再帰を含むプログラムとそのOR木

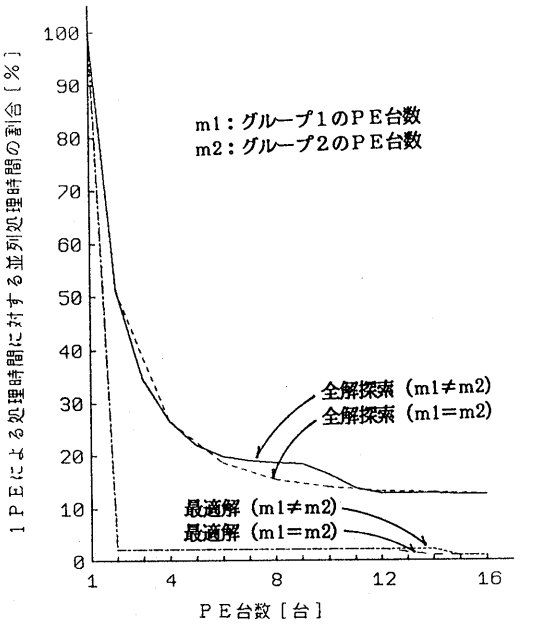


図10　PE台数による並列処理時間の減少度

み打ち探索を行なう。

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。従ってリーダ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なう。

4．　階層型挟み打ち探索手法の評価

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ない、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個の規則及び事実からなる。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なった。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に部分探索木を割当てるための時間及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全てグループ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。従って、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。PE16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、ＰＥ２台で５０倍，ＰＥ１６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、本手法を用いた場合得られる並列処理時間は最悪でも１ＰＥによる処理時間とほぼ同じであり、幅優先探索で見られる減速異常は原理的に生じない。ただし、リーダＰＥがＳＰを生成しブロードキャストする分のオーバヘッドは逐次の場合より増えてしまうがそれは僅かであると予想される。

次に左右から挟み打ちを行なう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かが明かではないため、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

本稿では、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なうので、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なうため、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ない、本手法の有効性及び実用性をチェックする計画である。

謝辞

本研究の遂行にあたり多くの御助力をいただいた本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、日頃から貴重な御協力をいただいた（株）東芝システムソフトウェア技術研究所中村英夫氏、本位田真一氏に感謝いたします。

参考文献

1）相田仁他：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）後藤厚宏他：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）増沢秀穂他：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 Computer，Vol.18，6，pp.93-108（1985）