記　号　処　理　61－2
（１９９１．　９．　１３）

# Ｐｒｏｌｏｇの並列処理方式

高橋　義造，宮本　直樹*
徳島大学・工学部・知能情報工学科，*現在　三洋電機（株）

Ｐｒｏｌｏｇの並列処理方式にはＯＲ並列とＡＮＤ並列の二つの方法があることが知られているが，本研究はＯＲ並列によるＰｕｒｅ　Ｐｒｏｌｏｇの並列処理システムを実現するための方式について検討を行ったものである．実際に，いくつかの異なる方法を用いたＰｒｏｌｏｇシステムを分散メモリ型マルチプロセッサ上に実装し，各種のプログラムを実行して並列処理効率を評価した．

# SEVERAL SCHEMES FOR PARALLEL EXECUTION OF PROLOG PROGRAMS

Yoshizo Takahashi and Naoki Miyamoto

Department of Information Science and Intelligent Systems
Faculty of Engineering, University of Tokushima
2-1 Minami Jyousanjima-cho, Tokushima 70, Japan

In parallel execution of Prolog programs, OR parallelism and AND parallelism can be used. In this paper different schemes for implementing OR parallel pure Prolog sytems are studied. By using these schemes several parallel Prolog systems were implementd on a distributed memory multiprocessor, and their efficiencies of parallel processing were evaluated.

## 1．はじめに

これまでにＰｒｏｌｏｇの並列処理方式については数多くの研究があり［1］，我々もかつてＰｕｒｅ　Ｐｒｏｌｏｇの並列処理を試みている．今回は負荷の均衡を図るため，以前とは異なる並列処理方式を研究した．本報告ではこの方式を以前の方式と比較して検討し，並列計算機を用いて性能評価を行った結果を報告する．

## 2．プログラム分散格納方式による並列Ｐｒｏｌｏｇシステム Ｅｎｏｌｏｇ

Ｐｒｏｌｏｇの並列処理方式は，ＯＲ並列処理とＡＮＤ並列処理に分けられる［2］．ＯＲ並列処理ではゴール節と頭部がマッチする節が多数ある時に，それらを並列にユニフィケーションを行う方式を取る．これに対してＡＮＤ並列処理では，節の本体を構成する複数のゴールの評価を並列に実行する方式をとる．ゴール間で変数の値を受け渡す必要があるので，ＡＮＤ処理処理はＯＲ並列処理より複雑になる．

我々の研究室ではＯＲ並列処理について研究し，先にプログラム分散方式の並列Ｐｕｒｅ　Ｐｒｏｌｏｇシステム Ｅｎｏｌｏｇを開発した［3］．このシステムではＰｒｏｌｏｇのプログラムを節単位で分割し，複数のプロセッサに分散して格納しておき，節ごとに並列にマッチングを行う方式である．節の後ろに；で区切って格納するプロセッサを指定することができる．指定しない場合には節の数が均等になるように自動的に割り当てる．例えば次のようなプログラムは図１のようにプロセッサに割り当てられる．

```
p(X,Y):- q(X,Z),q(Z,Y).   ;1   プロセッサ１に格納
q(a,b).                   ;2   プロセッサ２に格納
q(b,c).                   ;3   プロセッサ３に格納
```

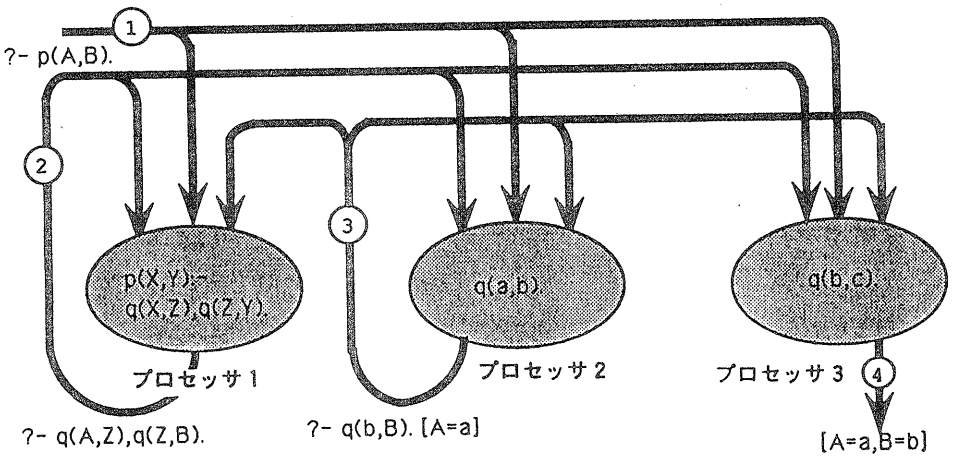


図１．Ｅｎｏｌｏｇのプログラム処理方式

プログラムの節は，予め内部コードに変換された後，プロセッサに格納される．プログラムを実行するには一つのゴール節をこれらのプロセッサに対して放送する．ゴール節には変数の値を入れるバスケットが付加されており，得られた変数のインスタンスをここに入れるようになっている．ゴール節を受け取ったプロセッサは，その節の頭部とマッチする先頭の述語を持つ節を，自分が格納しているプログラムの中から探し，ユニフィケーションを行う．ユニフィケーションが成功すると，その述語を得られたインスタンスに置き換えて得られる新しいゴール節を生成すると共にバスケットにそのインスタンスを入れ，それらを自分自身を含めた全てのプロセッサに放送する．このようにしてプロセッサ間を多数のゴール節が放送され，全ての述語が導出されると，バスケットの内容がホストコンピュータに回収され，答が表示される．

組み込み述語については各プロセッサが個別に処理を行うので，生成されるゴール節の最初の述語が組み込み述語の場合には，このゴール節は放送せず，そのプロセッサ内で処理される．ゴール節のユニフィケーションが成功し，これ以上評価する述語がなくなると実行は終了する．このように，すべてのゴール節の処理が終えるまでゴール節の放送とユニフィケーションが繰り返され，プログラムが処理される．

例えば図１のプロセッサにゴール節

？－　P（A，B）．

が与えられると，図１に示すようなゴール節が次々と生成され，最後にプロセッサ３から答A＝a，B＝cが出力される．図中の①，②などは生成されるゴール節の順番である．

この並列処理方式では一部のプロセッサに処理が集中し，負荷の不均衡が生ずる恐れがある．これは，プログラムを構成する節の中，特定の節でユニフィケーションが集中的に行われるためで，このような節が割り当てられたプロセッサの処理が他のプロセッサに較べて著しく多くなるために負荷の不均衡が起こり，並列処理効率が低下するのである．例えば次のような７クイーン問題のプログラムをこの方法で処理すると１番から１０番までのプロセッサのゴール生成回数は表１の様になり，著しい負荷のアンバランスが起ることがわかる．

```
queen(N,Q):- generate(N,L),place(L,[],Q).                          ;1
generate(0,[]).                                                    ;2
generate(N,[N|L]):- gt(N.0),sub(N,1,N1),generate(N1,L).            ;3
place([],Q,Q).                                                     ;4
place(QS,B,Q):- select(QS,QL,R),safe(B,QL),place(R,[QL|B],Q).      ;5
select([X|Y],X,Y).                                                 ;6
select([X|Y],Z,[X|W]):- select(Y,Z,W).                             ;7
safe(R,Q):-add(Q,1,Qi),sub(Q,1,Qd),safe1(R,Qi,Qd).                 ;8
safeel([],Qi,Qd).                                                  ;9
safe1([Q|R],Qi,Qd):- ne(Q,Qi),ne(Q,Qd),add(Qi,1,Qii),
                     sub(Qd,1,Qdd),safe1(R,Qii,Qdd).               ;10
```

表1　Ｅｎｏｌｏｇによる７クイーン問題のゴール生成回数
（総ゴール生成回数 23,253回，処理時間 165秒）

| ﾌﾟﾛｾｯｻ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ｺﾞｰﾙ生成回数 | 1 | 1 | 22 | 40 | 552 | 1345 | 1345 | 4035 | 551 | 15361 |

これを避けるには一つの節にユニフィケーションが集中しないようにプログラムを書き換え，できるだけユニフィケーションの回数が均一になるように節をプロセッサ台数に分割する必要がある．しかし実際にそれを完全に行うことは難しい．そこで次章に述べるような方法をとることになった．

## 3．タグ付きゴール節による負荷分散方式

プログラムをプロセッサに分散して格納すると，ユニフィケーションが並列に行われる可能性が増えるために並列処理の効果が上がることが期待される．しかし，先の例にみるようにプロセッサごとの負荷のアンバランスが大きくなるという欠点がでる．そこで負荷のアンバランスが起きないような方式を考える必要がある．このために各プロセッサにプログラム全体をもたせ，どの節でもユニフィケーションができるようにしておき，同じゴール節を受け取ったプロセッサが異なる節に対してユニフィケーションを行うようにする．このために，放送するゴール節にタグを付加し，各プロセッサは自分の番号とこのタグから計算される番号の節をユニフィケーションの対象とするようにした．

最初のゴール節のタグは０であるが，ユニフィケーションの結果生成されるゴール節は１，２，３，・・・と１づつふやして行き，プロセッサ数に達すると０に戻す．ただし，ゴール節の最初の述語が組み込み述語の場合には放送する必要がないのでタグの値は変化させない．ゴール節を受け取ったプロセッサは，このタグと自分のプロセッサ番号から次式によってユニフィケーションする節の番号を求める．

節番号＝（ｐｎｕｍ＋ｔａｇ）％Ｐ＋１＋ｉ＊Ｐ

ここに，Ｐはプロセッサ数であり，ｉは０，１，２，３．・・・の値をとる．例えばＰ＝３とすると，ｔａｇ＝２のゴール節は，プロセッサ１では１，４，７，・・の節と，プロセッサ２では２，５，８，・・・の節と，プロセッサ３では３，６，９，・・の節とユニフィケーションが行われる．このようにすると，あるゴール節は必ず１台の，また１台だけのプロセッサで処理されることになり，負荷がある程度均等に分散されることが期待される．実際にこの方式のＰｒｏｌｏｇシステムを作成し，先のプログラムを６３台のプロセッサで処理したときの各プロセッサの生成ゴール節を測定すると表２の様になり，表１と較べるとかなりの負荷分散が行われていることがわかる．また処理速度は約３倍になったが，稼働プロセッサ数が６倍になったにしては不満足である．

表２　タグ付きゴール節による７クイーン問題のゴール生成回数
（総ゴール生成回数 23,253回，処理時間 51秒）

| ﾌﾟﾛｾｯｻ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ｺﾞｰﾙ生成回数 | 269 | 282 | 355 | 190 | 23 | 392 | 833 | 175 | 12 | 31 | 141 | 451 |
| ﾌﾟﾛｾｯｻ番号 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| ｺﾞｰﾙ生成回数 | 714 | 776 | 612 | 163 | 67 | 9 | 8 | 40 | 77 | 179 | 280 | 534 |
| ﾌﾟﾛｾｯｻ番号 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| ｺﾞｰﾙ生成回数 | 646 | 735 | 888 | 738 | 719 | 550 | 350 | 133 | 99 | 64 | 31 | 6 |
| ﾌﾟﾛｾｯｻ番号 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| ｺﾞｰﾙ生成回数 | 6 | 13 | 18 | 53 | 63 | 90 | 107 | 200 | 244 | 285 | 318 | 553 |
| ﾌﾟﾛｾｯｻ番号 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| ｺﾞｰﾙ生成回数 | 600 | 643 | 651 | 792 | 860 | 902 | 896 | 736 | 711 | 707 | 669 | 485 |
| ﾌﾟﾛｾｯｻ番号 | 61 | 62 | 63 | | | | | | | | | |
| ｺﾞｰﾙ生成回数 | 447 | 345 | 287 | | | | | | | | | |

## ４．動的負荷分散方式のＮｅｏｌｏｇ

先の負荷分散方式では負荷はかなり分散されたものの，処理速度はまだ不十分であった．その主な理由はゴール節の放送に時間が費やされるためであると思われる．使用した並列計算機Ｃｏｒａ１６８Ｋは二進木結合網を用いた分散メモリ型マルチプロセッサであり，プロセッサによる中継によって放送を行う方式をとる．そのために放送に要する時間はバス結合等に較べるとかなり遅い．また７クイーンの例でもわかるように，２万回以上の放送が行われるので，この並列処理方式は分散メモリ型マルチプロセッサには適さないことがわかる．そこで放送を必要としない並列処理方式を考え，新しいＰｕｒｅ　ＰｒｏｌｏｇシステムＮｅｏｌｏｇを開発し実験を行った．

この方式はゴールをたらい回しにして，処理をしていないプロセッサに処理をさせる，いわゆる動的負荷分散法で，株わけ方式[1]と原理的には同じである．各プロセッサはプログラム全体を格納することは前章の方式と同じであるが，ゴール節にはタグのようなものは付加しない．最初のゴール節はは１番のプロセッサに送られるが，このプロセッサが生成するゴール節は近傍のプロセッサに順番に送られる．ゴール節を受け取ったプロセッサはｂｕｓｙ状態になり，ユニフィケーションを行う．この途中に次のゴール節が到着しても受け付けず，次のプロセッサに転送し，自分は処理を続ける．処理が終わって新しいゴール節が生成されると，これを近傍のプロセッサに送ってからｂｕｓｙ状態を終了する．この状態の時にゴール節が送られて来ると受け付けて処理する．

Ｃｏｒａ１６８Ｋではプロセッサは二進木状に結合されており，ノードのプロセッサには３方向に近傍プロセッサがある．今回のシステムでは生成したゴール節は３方向に順番

に送ることにした．またｂｕｓｙ状態の時に送られてきたゴール節は，送られてきた方向を除く２方向へ交互に転送する．葉の位置にあるプロセッサは送信方向は１方向しかないからｂｕｓｙの場合に到着したゴール節は送り返すことになる．このあたりの送信先を決める方式はもう少し洗練されたものにしなければならなかったかと思う．

この動的負荷分散方式では全てのプロセッサがｂｕｓｙになると，多数のゴール節がプロセッサの間をさまよう状態が起こる．この状態になると，通信時間が異常に増え，処理速度は大幅に減少する．例えば７クイーン問題では生成された23,253個のゴール節が，延べ115,629回，平均５回も転送され，最も多いものは６３回も転送されていることが測定された．　そこで，我々はゴール節の転送回数に制限を設けることにした．そのためにゴール節が１回転送されるごとにその回数をゴール節に付加しておき，転送回数が一定値に達したゴール節は，たとえそのプロセッサがｂｕｓｙであっても転送せず，一時バッファに保留しておき，ｂｕｓｙ状態が解ければユニフィケーションするようにした．

このようにして作成した並列Ｐｒｏｌｏｇシステムによる７クイーン問題の処理時間を測定した結果は１２秒であった．これを先のタグつきゴール節による負荷分散方式と較べると約４倍の，また元のシステムと較べると約１４倍の速度向上が得られたことになる．また負荷の分散状況を測定した結果は表３に示すようになり，かなり改善されていることがわかる．

表３　動的負荷分散方式による７クイーン問題のゴール生成回数
（総ゴール生成回数 23,253回，処理時間 12秒）

| ﾌﾟﾛｾｯｻ番号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ｺﾞｰﾙ生成回数 | 376 | 287 | 275 | 309 | 311 | 328 | 297 | 317 | 330 | 320 | 322 | 324 |
| ﾌﾟﾛｾｯｻ番号 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| ｺﾞｰﾙ生成回数 | 331 | 312 | 329 | 369 | 364 | 370 | 346 | 348 | 354 | 340 | 353 | 351 |
| ﾌﾟﾛｾｯｻ番号 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| ｺﾞｰﾙ生成回数 | 348 | 363 | 351 | 340 | 333 | 353 | 349 | 422 | 410 | 423 | 377 | 419 |
| ﾌﾟﾛｾｯｻ番号 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| ｺﾞｰﾙ生成回数 | 409 | 415 | 395 | 409 | 417 | 409 | 406 | 409 | 394 | 404 | 389 | 409 |
| ﾌﾟﾛｾｯｻ番号 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| ｺﾞｰﾙ生成回数 | 412 | 418 | 394 | 412 | 386 | 401 | 372 | 397 | 384 | 405 | 379 | 396 |
| ﾌﾟﾛｾｯｻ番号 | 61 | 62 | 63 | | | | | | | | | |
| ｺﾞｰﾙ生成回数 | 396 | 406 | 376 | | | | | | | | | |

## ５．並列処理効率の評価

Ｎｅｏｌｏｇの並列処理効率を評価するために，数種類のプログラムをプロセッサ数を

変えて実行し，速度向上率を測定した．その結果を表４に示す．

表４　処理時間と速度向上率の測定結果

| プログラム | | プロセッサ数 | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 3 | 7 | 15 | 31 | 63 |
| フィボナッチ数（15次） | 時間秒 | 195 | 168 | 163 | 165 | 164 | 164 |
| | 向上率 | 1.0 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| 200迄の素数 | 時間秒 | 265 | 227 | 219 | 219 | 219 | 219 |
| | 向上率 | 1.0 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| 7クイーン | 時間秒 | 339 | 194 | 93 | 47 | 23 | 12 |
| | 向上率 | 1.0 | 1.8 | 3.6 | 7.3 | 14.7 | 27.8 |
| 8クイーン | 時間秒 | 1624 | 939 | 452 | 224 | 110 | 54 |
| | 向上率 | 1.0 | 1.7 | 3.6 | 7.3 | 14.8 | 30.0 |
| 家系図探索 | 時間秒 | 38 | 17 | 8 | 4 | 2.4 | 1.5 |
| | 向上率 | 1.0 | 2.7 | 4.8 | 9.9 | 16.4 | 26.3 |
| ナップザック（6個） | 時間秒 | 212 | 107 | 107 | 101 | 100 | 100 |
| | 向上率 | 1.0 | 2.0 | 2.0 | 2.1 | 2.1 | 2.1 |
| クイックソート（50データ） | 時間秒 | 129 | 88 | 88 | 88 | 88 | 88 |
| | 向上率 | 1.0 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |

表４の結果から，Ｎｅｏｌｏｇの性能はプログラムに種類によってはっきり異なることが明かになった．並列処理効率の高いＮクイーンや家系図探索のプログラムは生成されるゴール節の数が非常に多い，すなわちユニフィケーション回数が非常に多いプログラムである．これに対し，他のプログラムでは生成されるゴール節が少なく，並列化による時間短縮が通信時間によって相殺されるために並列処理効果があがらない．

またＮｅｏｌｏｇをＥｎｏｌｏｇと比較してみると，次のようなことがわかる．Ｅｎｏｌｏｇではゴール節を放送することによって，節のサーチを並列に行うことができた．これに対してＮｅｏｌｏｇでは個々のゴール節のサーチが１台のプロセッサで逐次的に行われる．すなわちあるゴール節に対し，これとマッチングするプログラム中の節を逐次的に探索し，それが見つかる度にユニフィケーションを行って新しいゴール節を作って送信するので，その間他のプロセッサは待つことになり，並列処理が行われなことになる．系図探索問題などがＥｎｏｌｏｇではＮｅｏｌｏｇと同程度の時間で処理できるのはこのためであろう．逆にＮクイーン問題のような多数のゴール節が生成されるようなプログラムではＮｅｏｌｏｇの方が有利になる．

## ６．結論

以前に開発した並列Ｐｒｏｌｏｇシステムをベースに，タグ付きゴール節による負荷分散方式と動的負荷分散方式を採用したＰｒｏｌｏｇシステムを開発し，並列処理効率を実験的に評価した．Ｐｒｏｌｏｇの逐次的な処理を高速化するための研究は盛んであるが，

我々は効率の高い並列処理方式を開発することによって高速化を目指すための研究を行った．しかし今回の研究成果である，６３台のプロセッサで３０倍程度の速度向上率ではまだまだ不十分であり，ＯＲ並列だけでは限界があることが認識された．

## 参考文献


[1]柴山　潔：「並列記号処理」，コロナ社 (1991)

[2]Conery,L.S.:"Parallel Execution of Logic Programming Programs",Kluwer Academic Publishers (1987)

[3]Takahashi,Y.,Inoue,K.,Endo,T.:"Performance of Parallel Execution of Prolog Progorm by Goal Broadcasting on the Binary-Tree Parallel Computer", Proc. of INternational Computer Symposium, Taipei, pp. 743-748 (1988)