4Q-9

並列化Prolog処理系「lonli+」(2)
— 並列化変換 —

車谷　博之*　　加藤　整**　　広瀬　正*
*(株)日立製作所　システム開発研究所
**(株)日立マイクロコンピュータエンジニアリング

1．はじめに

現在、ワークステーションが普及しており、近い将来、市販のマイクロプロセッサを用いたマルチプロセッサシステムが低コストで利用可能となると思われる。Prolog高速化の一手段として、このようなシステムの下でのPrologＯＲ並列処理を検討している。この検討に際して、以下の方針を立てた。

(1)従来Prologとの上位互換性を保持すること。

(2)従来処理系とからの移行性を重視。

過去Prolog処理系を研究開発しており、この処理系からの移行を容易にする。

(3)並列実行指示を、ユーザ指示可能かつ自動並列化可能とすること。

これらの方針を満たすため、並列実行制御機能をProlog処理系の組込述語(並列実行制御述語という)として実現する方法を採った。従来の処理系に並列実行制御述語を加えることで、上位互換性の保持、従来処理系からの移行の容易化、ユーザ指示可能な並列化を行う。また、Prologソースプログラムを解析して並列実行制御述語を挿入するツール(並列化変換系)を提供することで、自動並列化を行う。

2．並列実行制御述語

(1)divide述語

プロセスを生成する(図1)。

本述語実行プロセスは:「divide≡fail」と解釈し、実行を続ける。

生成されたプロセスは「divide ≡ '!'」と解釈し、実行を続ける。

プロセスはdivideにバックトラックした時点で消滅する。

プロセス①は述語ｐのdivide述語を実行するとプロセス②を生成し、プロセス②はfailと解釈するのでバックトラックし、ｐの第２クローズを実行し、述語ｑを呼び出す。

(2)dfg述語

本述語はプロセスの深さ優先探索の順序(Prolog逐次実行順)を回復し、従来Prolog処理系との互換性を確保するために用いる。

深さ優先順で前のプロセスすべての消滅を待つ。この待合せにより深さ優先探索の順序を回復する。「dfg≡true」と解釈する。dfgはdepth_first_gateの略語であり、バックトラックすると必ず失敗する述語である。プロセス③はdfg述語を実行するとプロセス①及びプロセス②の消滅まで待つ。

3．並列実行制御述語実行方式

並列実行制御述語＋Prologプログラムを日立ワークステーション2050/32(HI-UX、シングルプロセッサ)で実現した。

(1)divide述語の実行

fork機能を用いて実現する。

(2)dfg述語、para述語の実行

プロセスの深さ優先順序を共有メモリに保持する。また、プロセス間の待合せは、メッセージキュー機能を用いて実現する。

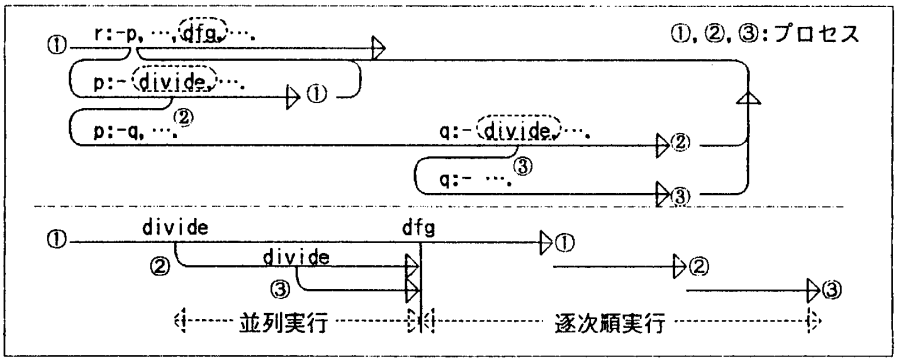

図1．並列実行制御述語(divide,dfg)の動作

An Parallel Prolog processor LONLI+ -from Prolog programs to Parallel executable programs-
Hiroyuki KURUMATANI*,Oasamu KATO**,Tadashi HIROSE*
*Systems Development Laboratory,Hitachi Ltd.**HITACHI Micro Computer Engineering,Inc.

4．並列化変換方式

2以上のクローズからなる述語の各クローズを並列に実行する。i番目と(i+1)番目のクローズを並列に実行するため、i番目のクローズのボディ先頭にdivide述語挿入する。

```
p(X):-a,b,!,c(X).          ～[1]
p(X):-b.                   ～[2]
a:-slow,fail.              ～[3]
a:-slow4.                  ～[4]
b:-slow5.                  ～[5]
b:-slow6.                  ～[6]
c(X):-slow7(X).            ～[7]
c(X):-slow8(X).            ～[8]
slow:-slow1.               ～[9]
slow:-slow2.               ～[10]
slow:-slow3.               ～[11]

?-p(X),write(X),….         ～[12]
```

図２．Prologプログラム例

```
p'(X):-a',b,!,c'(X).       ～[1']
p'(X):-b'.                 ～[2']
a':-divide,slow',fail.     ～[3']
a':-slow4.                 ～[4']
b:-slow5.                  ～[5]
b:-slow6.                  ～[6]
b':-divide,slow5.          ～[5']
b':-slow6.                 ～[6']
c'(X):-divide,slow7(X).    ～[7']
c'(X):-slow8(X).           ～[8']
slow':-divide,slow1.       ～[9']
slow':-divide,slow2.       ～[10']
slow':-slow3.              ～[11']

?-p'(X),dfg,write(X),….    ～[12']
```

図３．図２のプログラムの並列化変換結果

ＯＲ並列実行は、Prologのバックトラックによる別解探索を、解と別解を同時に並列実行する方法である。ところが、Prolog言語には、カット述語がありこの述語を実行すると別解探索を中止する。カット述語実行によって刈られる探索パス時間が無駄になるリスクを犯しても、Prologのカット述語影響範囲を並列実行するアプローチ[1]もある。しかし、我々は、並列実行に相応しい問題は、カット述語の影響範囲を除いても、十分に並列度があろうと考え、この範囲を並列実行から除くことにした。

図２のPrologソースプログラムを解析し、divide述語やdfg述語を挿入したものを図３示す。Prologソースプログラムから、カット述語実行の影響範囲、すなわち別解探索を中止する範囲を避けて、並列実行を指示する述語divideを挿入する。ここで、述語ａ、ｂ、ｃを並列実行する。しかし、クローズ[1]のカット述語('!')実行の影響範囲は、クローズ[1]の述語呼出しｂである。ここで、クローズ[2]の述語呼び出しbの実行は、この影響範囲外であるので、クローズ[2]の述語呼び出しbのために述語bを複写し名称を変更し、述語b'とする(クローズ[5']とクローズ[6'])。また、クローズ[3]は、fail述語を実行するため、クローズ[3]のボディでdivideを実行しても、生成したプロセスはすべてdivideにバックトラックした時点で消滅し、クローズ[1]のカット述語を実行しない。従って、述語a、述語slowは並列実行可能であり、divide述語を挿入し、クローズ[3']、[9']、[10']とする。さらに、クローズ[1]の述語呼び出しcは、クローズ[1]のカット述語実行影響範囲外であり、述語cを並列実行するために、divide述語を挿入し、クローズ[7']とする。

また、副作用を起こす述語(assert、retract、readなど)の直前にdfg述語を挿入する。ここで、ユーザが副作用の必要がないと判断した場合には、このdfg述語を取り除き効率良く実行することが可能である。

5．おわりに

並列実行制御述語は言語Ｃで、1.2k行程度で実現した。また、並列化変換系は言語Prologで約500行である。本処理系は、シングルプロセッサで動作するが、fork、共有メモリ、セマフォ、メッセージキュー機能を持つマルチプロセッサ用のＯＳが実働しているマルチプロセッサシステムには容易に移植可能である。

参考文献

1) B.Hausman,A.Ciepielewski,A.Calderwood," Cut and Side-Effects in Or-Parallel Prolog",Proc. of FGCS '88,pp831-840.