論理型言語の並列プログラミング

6D-6

－ 並列度向上支援ツール －

辰口 和保　村岡 洋一

(早稲田大学理工学部)

1．はじめに

論理型言語を並列処理する並列計算機の研究が、近年盛んに行われている。しかし、そのような並列計算機上で、論理型言語で書かれた応用プログラムがどのような挙動を示すか、さらにはそのような応用プログラムの作り方「並列プログラミング手法」の研究は、あまりなされていない。我々は先に、Prologの動的解析ツールを作成し、OR並列Prologの挙動を測定した[1]。今回は、このツールを並列プログラミング手法研究用に発展させた「並列度向上支援ツール」について報告する。

2．並列Prolog

AND/OR並列による動的並列度の測定について、文献[2]ではレベルという考えが導入されている。1レベルは、ユニフィケーションが実行され、次のゴールが生成されるまでをいう。あるレベルでの並列度は、そのレベルで実行される（成否を問わない）ユニフィケーションの数である。また、引数間のユニフィケーション並列については、文献[3]で未束縛の変数によって引数をクラスタに分割し、クラスタごとに並列にユニフィケーションを行う手法が提案されている。ユニフィケーション並列による並列度は、このクラスタ数と考えることができる。

これらの並列性を活かして実行される並列Prologの仕様は、pure Prologにストリーム AND並列実行を可能とするためのread only annotationを付加したものとする。OR並列Prologをもとにしているので、commitは行わず全解探索をする。

本ツールで「向上」される並列度はAND/OR並列によるもので、引数間ユニフィケーション並列の並列度は、動的に測定するにとどめ、必要に応じて並列度向上操作の参考にする。

3．並列度向上支援ツール

本ツールは、

- ユニフィケーションの部分実行
- unfold/fold 変換によるプログラム変換
- 他の一般的最適化技法

を並列度を向上させるように使用するプログラム変換ツールである。その全体像を、fig.1に示す。ツールは、測定系と変換系の２つの部分より成る。

・測定系
測定系は、静的測定系と動的測定系の２種がある。

i)静的測定系
プログラムを頭から読込み、次のようなデータを収集し動的測定系の読込める形にプログラムを変換する。

OR関係節数
AND関係節数
述語の静的呼びだし回数
述語の独立性
初期の引数クラスタ

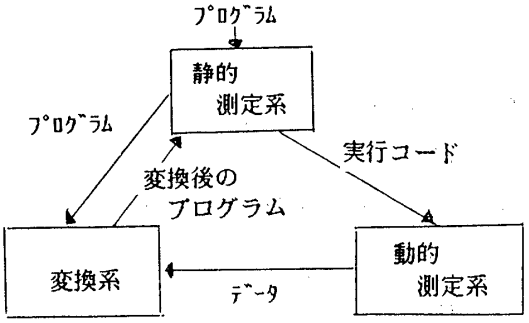

fig.1 並列度向上支援ツール

ii)動的測定系
文献[4]のConcurrent Prologインタプリタをもとに作成した測定用インタプリタで、次のようなデータを収集する。

推論木の最大幅（最大並列度）とそのレベル
推論木の高さ
推論木の各レベルの幅
各レベルでのユニフィケーション並列度の総和
節のheadとbodyの成功・失敗の数
組込み述語の成功・失敗の数
述語の呼びだし状況

・変換系
変換系としては、現在３つがある。
i)インライン展開ツール
ユニフィケーションの部分実行によりインライン展開を行う。

ii)簡易appendオプティマイザ
文献[5]のappendを多用したプログラムをappendを用いない、より効率的なプログラムに変換するツール。

iii)一般プログラム変換エディタ
unfold/fold変換によるプログラム変換を支援するエディタ。

本ツールは、測定系によりプログラムの挙動を測定し、それをもとに変換系を用いて並列度の向上操作を行う。変換系を使用するにあたって注目する点は、

- 呼びだし回数の多い述語の並列度向上
- 直線的述語定義の除去
- 再帰プログラムの展開による並列度の向上

である。変換ツールの使用順は、unfold/fold変換により仕事の総量が減少したプログラムにインライン展開を施すのを繰返すことにする。

Parallel Programming in Logic Programming Language

Kazuyasu TATSUGUCHI,Yoichi MURAOKA

WASEDA Univ.

```
reverse([],[]).
reverse([A|X],Y):-reverse(X?,Z),append(Z?,[A?],Y).

maximum level = 11
maximum width = 22 at 11
total number of width = 132

reverse 1: call_num = 11
reverse 2: call_num = 11
append 1 : call_num = 55
append 2 : call_num = 55
```

fig.2 reverse (10要素逆転)

```
reverse([A],[A]).
reverse([A1,A2|X],Y):-rev(X?,[A?,A1?],Y).
rev([],Y,Y).
rev([A],V,[A|V]).
rev([A1,A|X],V,Y):-rev(X?,[A?,A1?|V?],Y).

maximum level = 6
maximum width =3 at 6
total number of width = 17

reverse 1: call_num = 1
reverse 1: call_num = 1
rev 1    : call_num = 5
rev 2    : call_num = 5
rev 3    : call_num = 5
```

fig.3 reverse 変換後 (10要素逆転)

4. 並列度向上の例

本ツールによる並列度向上の例をfig.2～fig.5に示す。

fig.2は、appendを用いたreverseの例である。これにappendオプティマイザを適用し、さらにインライン展開を施すとfig.3になる。10要素のリストの逆転を行ったデータを付してある。並列度の減少は、仕事の総量が激減したためで、appendオプティマイズのみの場合は、最大並列度は2である。

fig.4は、行列の掛算のプログラムであり、直線的述語定義の除去、及びインライン展開を各述語に施したものがfig.5ある。データは、3次の行列の掛算を行った場合である。

toy programに対する例であるので、並列度の向上はそれほど大きくない。大規模な応用プログラムでは、各所での向上効果が総合され、大きな並列度の向上が見込まれる。

```
mm([],_,[]).
mm([Xi|Mx],My,[Zi|Mz2]):-
     vm(Xi?,My?,Zi),mm(Mx?,My?,Mi2).
vm(_,[],[]).
vm(Xi,[Yj|My],[Zij|Mi2]):-
     ip(Xi?,Yj?,Zij),vm(Xi?,My?,Mi2).
ip(Xi,Yj,Zij):-ip2(Xi?,Yj?,0,Zij).
ip2([Xik|Xi],[Yjk|Yj],Sum,Zij):-
     Sum2 is Xik*Yjk+Sum,ip2(Xi?,Yj?,Sum2?,Zij).
ip2([],[],Zij,Zij).

maximum level = 11
maximum width = 25 at 7
total number of width = 140

mm 1 : call_num = 4
mm 2 : call_num = 4
vm 1 : call_num = 12
vm 2 : call_num = 12
ip 1 : call_num = 9
ip2 1: call_num = 36
ip2 2: call_num = 36
```

fig.4 行列の掛算 (3次行列)

```
mm([],_,[]).
mm([Xi],My,[Zi]):-vm(Xi?,My?,Zi).
mm([Xi,Xi2|Mx],My,[Zi,Zi2|Mx2]):-
     vm(Xi?,My?,Zi),
     vm(Xi2?,My?,Zi2),
     mm(Mx?,My?,Mz2).
vm(_,[],[]).
vm(Xi,[Yj],[Zij]):-ip2(Xi?,Yj?,0,Zij).
vm(Xi,[Yj,Yj2|My],[Zij,Zij2|Mi2]):-
     ip2(Xi?,Yj?,0,Zij),
     ip2(Xi?,Yj2?,0,Zij2),
     vm(Xi?,My?,Mi2).
ip2([Xik,Xik2|Xi],[Yjk,Yjk2|Yj],Sum,Zij):-
     Sum2 is Xik2*Yjk2+Xik*Yjk+Sum,
     ip2(Xi?,Yj?,Sum2?,Zij).
ip2([Xik],[Yjk],Sum,Sum2):-
     Sum2 is Xik*Yjk+Sum.
ip2([],[],Sum,Sum).

maximum level = 7
maximum width = 31 at 4
total number of width = 96

mm 1 : call_num = 2
mm 2 : call_num = 2
mm 3 : call_num = 2
vm 1 : call_num = 6
vm 2 : call_num = 6
vm 3 : call_num = 6
ip2 1: call_num = 18
ip2 2: call_num = 18
ip2 3: call_num = 18
```

fig.5 行列の掛算 変換後 (3次行列)

5. おわりに

論理型言語によるプログラムの並列度を向上させる支援をするツールについて報告を行った。既存の効率化技法を組織的に用いることで、並列度を向上させることができる。しかし、さらなる並列度の向上には新しい手法の開発が望まれる。また、プログラムの動的解析もプログラムを実行しなくては分らないのでは不便である。さらに言語仕様も実装上問題がある。これらの点を当面の研究課題として、「並列プログラミング手法」の確立へ向け、研究を進めたい。

6. 参考文献

[1] 辰口 村岡：" Prologマシンのアーキテクチャに関する一提案"，情報処理第32回全国大会.

[2] 尾内 他 ：" 逐次型Prologプログラムの解析"，Logic Programming Conference '84. Tokyo,1984.

[3] 山口 他 ：" 定理証明プログラムにおける単一化計算の並列処理について"，信学論 J67-D,No.3,pp.289-296,(1984.3).

[4] E.Y.Shapiro:"A Subset of Concurrent Prolog and Its Interpreter",ICOT TR-003(1983.1).

[5] 玉木、佐藤：" appendオプティマイザについて"，Logic Programming Conference '84. Tokyo,1984.