並列論理型言語のコンパイル技法

3Q-3

上田 和紀
(日本電気(株) C&C システム研究所)

0. はじめに

最近1年間ほど, Concurrent Prolog[1] (CP)のコンパイル技法の研究を行なってきた。またそれと並行してCPの言語仕様の問題点を考察・検討してきた。それらの成果の概要を報告する。

1. コンパイル技法

並列論理型言語としてCPを取りあげ, 以下の項目を中心に検討を行なった。対象としたマシンは通常の汎用機である。汎用機上での実現法の研究は, 速度や移植性の点で当面最も有利であること, 近未来の並列マシンにおいて各プロセサが多プロセスを扱う方法を提供すること等の理由できわめて重要である。

1-a) ストリーム並列の効率的実現

並列論理型言語を実用的な汎用言語とするためには, ストリーム並列[2]を効率的に実現しなければならない。特に多対1のプロセス間通信の際必要なストリームの併合を効率よく実現する必要がある。筆者らは, n本のストリーム(静的にはリスト)を併合する述語を解析し, コンパイル技法の工夫によってO(1)の遅れをもつ併合器が実現できることを示した[3]。この併合器は, 入力ストリーム数nの動的増減を許す併合器に(O(1)の遅れを保ったままで)拡張できる。実用上は, 併合器は組込述語として定義するのがよい。

さらに3)では, 配列をゴールとして実現すると, O(1)の時間で読み書きができることを示した。ここでは配列は

```
array(N, S)   %Nは配列の大きさ
```

というゴールであり, 配列の利用者はストリームSを通じて, read(添字, 変数)・write(添字, 値)といったメッセージを送りこむ。このようにストリームをインターフェースとして用いると、(要素の出し入れのできる)二分木や重つなぎ並びのような, Prologでは実現のむずかしかった動的データ構造も容易に実現できる。

このような, ストリームを多用するプログラミングスタイルが実用的であるためには, ストリーム通信の効率が非常に高くなければならない。文字列ストリームについては4)でべたづめ表現法を提案したが, 一般のストリームの効率的実現についても明るい見通しを得ている。

1-b) 実用処理系

CPのプログラムをDEC-10 Prologのプログラムに変換するコンパイラを作成した[5]。DEC-10 Prologはコンパイラを持っているから, CPプログラムは結局機械語になって走る。

速度は予想以上に高く, モード宣言を与えたappendプログラムで11.7 kLIPSを得た(DEC2060)。ちなみにDEC-10 Prologでのappendの実行速度は, コンパイラで42 kLIPS, インタプリタで2.7 kLIPSであった。高効率の源は, CPのユニファイア(Prologプログラムとして定義する必要がある)を可能な限り最適化したこと, およびCPのtail recursiveなプログラムがPrologのtail recursiveなプログラムに変換されるようにしたことにある。Tail recursiveなプログラムの最適化は, ストリームを多用するプロ

グラムにとってきわめて重要である。

記述言語とターゲット言語をPrologにしたことにより，処理系は短期間のうちに完成した。これにはソース・ターゲット・記述言語間の類似性と，それに伴う実行時支援作成作業の軽減が大きく寄与している。実現を容易にするために「失敗」の概念を導入せず，また動的待合せを用いたりもしているが，実用上の不都合は今のところない。

2. CPの言語仕様の問題とその解決

CPの言語仕様を，並列実行という観点から詳細に検討した[6]。以下にその主な結果を要約する。

a) 述語頭部の単一化およびガード部の実行は，すべて並列に行なうべきである。

b) ガード部の局所環境[7~9]と大域環境とが矛盾する場合，その矛盾の発生のしかたによって，commit前に検出しなければならない場合と，(実現の困難さ故) commit後の検出を許さなければならない場合とがある。

c) XとF(Y?)との単一化の意味を，XとF(Z)，ZとY? というふたつの単一化に分けてとらえてはいけない。

aはCPの場合，実現上の負担を重くする。bとcは実用上の不都合は少ないように見えるが，意味記述が複雑化し，プログラム変換のような操作をむずかしくする。

上記の問題点はすべて，CPのread-only annotationと多環境の存在に起因している。そこで，それらを廃止した単純な言語を提案した[10]。CPとの主要な相違は次の一点である。CPでは，commit前に，ゴール側の変数を他のゴール側変数や非変数項と単一化する場合，そのユニファイアを局所的に保存し，commit時に外に公開する。新しい言語では，そのような単一化は，自らユニファイアを生成せずに成功するようになるまで中断させる。この変更により，必要な記述力を保ったままで，CPの種々の問題点を解決し，CPよりもPARLOG[11]よりも単純な言語とすることができた。

なお，本研究は第5世代コンピュータプロジェクトの一環として行なった。

参考文献

1) Shapiro, E., A Subset of Concurrent Prolog and Its Interpreter, ICOT Tech. Report TR-003 (1983).

2) Conery, J. and Kibler, D., Parallel Interpretation of Logic Programs, Proc. 1981 Conf. on Functional Programming Languages and Computer Architecture, pp. 163-170 (1981).

3) Ueda, K. and Chikayama, T., Efficient Stream/Array Processing in Logic Programming Languages, Proc. FGCS '84, pp. 317-326 (1984).

4) 上田他，論理型言語による文字列処理の記述について，情報処理学会記号処理研究会資料 26-1 (1983).

5) 上田・近山，並列論理型言語の実用処理系，日本ソフトウェア科学会第1回大会 3D-5 (1984).

6) Ueda, K., Concurrent Prolog Re-Examined, to appear as ICOT TR.

7~9) 宮崎他，佐藤他[8]，田中他[9]，Concurrent Prologのシーケンシャルインプリメンテーション——Shallow binding (Deep binding[8], Copy[9]) 方式による多環境の実現——日本ソフトウェア科学会第1回大会 3D-2~4 (1984).

10) Ueda, K., Guarded Horn Clauses, to appear as ICOT TR.

11) Clark, K. and Gregory S., PARLOG: Parallel Programming in Logic, Research Report DOC 84/4, Dept. of Computing, Imperial College (1984).