ＡＩＰ－Ｐｒｏｌｏｇ：最適化の概要

7P-8

山田朝彦　岡本利夫
(株)東芝

0．はじめに

ＡＩＰ（ＡＩプロセッサ）用のPrologコンパイラは、D.H.D.WarrenによるWAM(Warren's abstract machine - Warrenの仮想マシン）に基づいて開発された。当初ＡＩＰ用コンパイラに最適化アルゴリズムは組み込まれていなかったが、実行速度を向上させるために次のような最適化を行った。

＊レジスタ割り付け及びオブジェクトコードの最適化
＊インデキシングの強化
＊組込述語のオブジェクト化とインライン展開

以下、上の３点について簡単に解説する。

1．レジスタ割り付け及びオブジェクトコードの最適化

当初開発したコンパイラでは、述語のヘッド部に現れるテンポラリ変数は、ボディ部に現れるかどうかにかかわらず必ずテンポラリレジスタに格納され、ボディ部の第１ゴールの引数として現れた際には、再び引数レジスタに格納され直されていた。つまり、テンポラリレジスタに移す必要がないにもかかわらず移されているというケースが数多くあった。これらは１命令ないし２命令をnop にできるものであるが、テンポラリレジスタの割り付けに工夫をすれば、命令をさらに減らせる場合も少なくない。

既存コンパイラによる生成コード量との比較をいくつかのプログラムで行ったところ、上記の最適化を行った場合のコード生成減少率は０％から１８％であり、平均すると約１０％である。ちなみに、８クィーンでは１０％の減少があった。

2．インデキシングの強化

2．1．WAM におけるインデキシングの問題点

WAM ではswitch on term命令により第１引数のタグによるマルチウェイジャンプを行って、節のマッチングの効率化を図っている。しかし、以下のようなプログラムではswitch on termは効果を発揮しない。

```
not_member( _,[]) :- !.
not_member(A,[B | L]) :- A=¥=B,not_member(A,L).
```

これが第１の問題点である。この場合は第２引数によりインデキシングを行えば効果がある（[]はコンスタントとして扱われているからである）。

第２の問題点としては、第１引数がコンスタント、ストラクチャの場合はswitch on termの飛び先でさらにそれぞれswitch on constant, switch on structure によりマッチングの効率化が図られるが、リストの場合はリニアサーチになり、変数の場合は該当する述語の節全てとのマッチングを行うことになるため、非常に多くの実行時間を要することになる。

2．2．多重インデキシングによる改造

第１の問題点に対しては、次のような方法による解決を試みた。コンパイルの時点で第１引数から第５引数までの、コンスタント、リスト、ストラクチャの数をそれぞれ評価し、インデキシングに最適な引数を選択するようにした。あとに述べるようにリストの場合でもリニアサーチでなくなるようにしてあるが、やはりリストの場合よりもコンスタント、ストラクチャの場合のほうが効率が良いので、引数選択を行う際にコンスタントとストラクチャにはリストよりも重い重み付けをして評価を行っている。

第２の問題点の解決法が多重インデキシング（厳密にいえば２重インデキシング）である。インデキシング引数がリストの場合は従来は既に述べたようにswitch on termの飛び先ではリニアサーチになっていたが、今回の改造ではリストのcar とインデキシング引数以外の引数を上に述べたのと同様の方法で評価しインデキシングに適当な方を選択するようにした。インデキシング引数が変数である場合は非常に効率が悪いが、この場合もリストの場合とほぼ同様にインデキシング引数以外の引数でインデキシングに効果のあるものを選び再度インデキシングを行うようにした。

これを図示すると図１のようになる。

2．3．本改造の問題点

インデキシングは、レジスタ割り付けとオブジェクトコードの最適化とは異なり、常に実行速度の向上に貢献するとは限らないという問題点をかかえている。それはWarren自身が考えたswitch on termの持っている問題でもある。その問題は、インデキシング引数として変数が入ってきた場合にはインデキシングを行わないのに比べてswitch on termを実行する分だけ実行時間を多く要することである。しかし、第１引数が変数である場合は実際は多くないのでさほど問題にはならないと考えられる。勿論、インデキシング引数が変数以外の場合には実行時間は短縮され、一般の場合に解を得るまでに要する時間の平均値も短縮される。

今回の改造では、この現象は増幅される。本処理系では一般には２つのインデキシング引数を持つが、その２つの引数がともに変数であった場合には、インデキシングを全く行わない場合と比較すれば１回の述語呼び出しにつきswitch on term２回分の実行時間が多くかかることになる。また、１回目のインデキシング引数がリストの場合、２回目のインデキシング引数が変数の場合には１回の述語呼び

Abstract of Optimization in AIP-Prolog
Asahiko YAMADA,Toshio OKAMOTO
TOSHIBA Corp.

出しにつきswitch on termが１回多く行われることになる。これらの場合も、改造前のワーストケースと同様に多くはおこらないので、大きな問題ではない。その他の場合については、１回目のインデキシング引数がコンスタントとストラクチャのときには改造前と変わらず、リストと変数の場合には処理時間は短縮される。解を得るまでに要する時間の平均値も短縮される。

インデキシングの効果があらわれるかどうかは、実際に実行されるまでは結果が出ないだけに、コンパイラをさらに改良する必要がある。インデキシング引数を選択する際の評価式の再検討などがおもな課題である。

３．組込述語のオブジェクト化とインライン展開

本処理系はインタプリタをエンジニアリングワークステーションAS3000上にもちコンパイルドコードをＡＩＰ上で実行するという構成になっている。すなわち、コンパイルされた述語中でコンパイルされていない述語が呼ばれた場合には、ＡＩＰからAS3000に制御を移さなければならず、そのための制御変換処理を行わなければならない。この変換処理はかなり重いので、組込述語のうち多用されるis、比較演算などのいくつかをＡＩＰ上で動作するようにした。これを我々は組込述語のオブジェクト化と称している。

組込述語のオブジェクト化によりAS3000とＡＩＰの間の通信は減少したが、これらの組込述語中に算術式があらわれた場合その算術式を評価するために数多くのタグチェックが必要なため、ＡＩＰでの実行ステップ中における組込述語の実行ステップの占める割合が非常に高いことがわかった。

組込述語の処理をさらに軽減するために組込述語のインライン展開を行った。インライン展開により、実行時に通過するチェックルーチンが大幅に減少するので、処理速度の向上が確認できた。

４．おわりに

以上ＡＩＰ－Ｐｒｏｌｏｇにおける最適化法について述べた。今後は、最適化をさらに改良し、また、インタプリタをＡＩＰに移植するなどして処理の高速化を図っていきたい。

参考文献

［１］ D.H.D.Warren, "An abstract prolog instructio-n set" TR309,SRI,October,1983

［２］ 斎藤 他：“Ｐｒｏｌｏｇを６６７ＫＬＩＰＳで実行するＲＩＳＣ風プロセサ” 日経エレクトロニクス（Ｎｏ．４３６），１９８７

［３］ 岡本 他：“バックエンド方式によりプロセッサを結合したＡＩワークステーション「ＷＩＮＥ」の構成法” 電気学会論文誌Ｃ（電子・情報・システム部門誌），１９８８

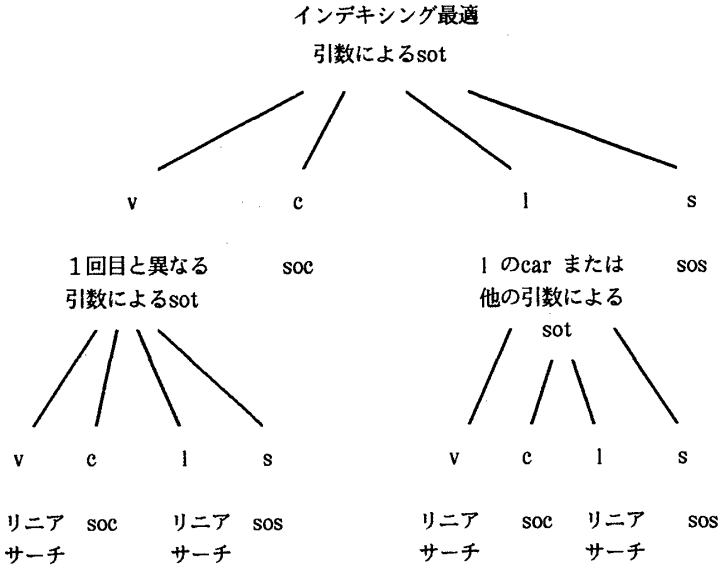

図 1