ICOT Technical Memorandum: TM-0284

TM-0284

小型化版 CHI の SUPLOG コンパイラにおける最適化技法

新　　淳，小長谷明彦
（日本電気）

March, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191－5
Telex ICOT J32964

Institute for New Generation Computer Technology

小型化版ＣＨＩの ＳＵＰＬＯＧコンパイラにおける 最適化技法

新　潟　・　小長谷　明彦
日本電気（株）　Ｃ＆Ｃシステム研究所

１．はじめに

筆者らは第５世代計算機のプロジェクトの一環として、論理型言語の高速実行を目指したバックエンド型推論マシンＣＨＩ（以下ＣＨＩ１号機と略す）[1]を小型化、改良した小型化版ＣＨＩを開発中である[2]。本稿では、小型化版ＣＨＩのシステム記述言語ＳＵＰＬＯＧのコンパイラにおける最適化技法のうち、型判定述語や算術演算述語といった組込述語を多方向分岐命令を用いてインライン展開する技法ならびに遅延インデクシング命令を用いてリスト処理を高速化する技法について述べる。これらの技法を用いることにより、小型化版ＣＨＩにおけるappend/3の予測性能は約400K LIPSとなる。

２．Ｐｒｏｌｏｇコンパイラにおける最適化技法

Prologのコンパイル方式として、D.H. Warrenの提唱した抽象マシン[3]（以下WAMと呼ぶ）を対象とするのが一般的になりつつあるが、このコンパイル方式においては、以下のような最適化が可能である[4]。

①ローカルスタックのトリミング

本体部の最後のゴールの呼出し時には、環境フレームを捨てることが可能であるが、WAMではこれを更に拡張して本体部の各々のゴールの呼出し時に環境フレームをトリミングすることによってローカルスタックの消費量を抑えることができる。

②インデクシング

手続き単位に述語をコンパイルする場合、各々の節の第一引数に関して解析を行うことによって候補節を絞り込むことが可能である。この技法をインデクシングと呼ぶ。

③不要なレジスタ転送命令の削除。

一時変数に対しては、これをレジスタに適切に割当てることによって冗長なレジスタ転送命令を削除することが可能である。

ＣＨＩ１号機では①から③の最適化を行ったが、性能向上の要因は選択肢フレームや環境フレームの生成をできる限り少なくすること、述語呼出しを高速化することの２点にあることがわかった[5]。このような観点から小型化版ＣＨＩでは更に以下の最適化を導入した。

④組込述語のインライン展開。

実行時にレジスタを破壊しない組込述語（例えば型判定述語、算術演算述語）は機械語命令列に展開することによって述語呼出し命令（場合によっては環境フレーム生成命令）を削除することができる。

⑤拡張インデクシングの導入。

インデクシングは、通常節の頭部の第１引数に対して行われるが、更に本体部の第１ゴールが型判定述語（例えばvar/1）である場合に、これを考慮した拡張インデクシングを行うことによって選択肢フレームの生成を減らすことができる。

⑥遅延分岐命令の導入によるループの高速化

Prologではリストがニルになるまでループを行うような処理が頻用されるが、このようなプログラムを対象とした遅延分岐命令を導入してコンパイルを行うことによって高速なループを実現することができる。

以下、④から⑥の最適化を中心にこれらの最適化を実現するために必要な機械語命令とプログラムの展開形について紹介する。

３．組込述語のインライン展開

３．１．型判定述語の展開

述語var/1を通常の述語と同等に扱って

```
f(X,Y) :- var(Y).g(Y,X).
```

をコンパイルすると、次のようなコードを得る。

```
        allocate                2
        get_permanent_variable  A0,Y0
        get_permanent_variable  A1,Y1
        put_permanent_value     A0,Y1
        call_built_in           var/1
        put_permanent_value     A0,Y0
        put_permanent_value     A1,Y1
        deallocate
        execute                 g/2
```

小型化版ＣＨＩでは、var/1を多方向分岐命令tagbranchを用いてインライン展開することによって述語呼出し、並びに環境フレームの作成を省略できる。更にレジスタ割付けを適切に行うことにより最終的に次のようなコードを生成する。

```
        deref   g0,A1       % dereference A1
        tagbranch(g0,tagbr$var$2way)
        jump    $1          % variable case
        fail                % non-var cases
$1:     put_register_value  A0,A3
```

Optimization Technique in SUPLOG Compiler for CHI Compact Version

Atsushi Atarashi, Akihiko Konagaya
NEC Corporation

```
        put_register_value    g0,A0
        put_register_value    A3,g0
        execute               g/1
```

デリファレンス命令derefはレジスタを２つ指定し、一方のレジスタの内容をデリファレンスして他方のレジスタにセットする命令である。上の例では、引数レジスタA1をデリファレンスし汎用レジスタg0にセットする。

タグ分岐命令tagbranchはレジスタとn方向分岐パターン(n≦8)を指定し、直後にnケの命令を並べて使用する。タグ分岐命令は指定されたレジスタのタグを調べ、タグ分岐命令+i番目(i≦8)の命令を実行する。上の例ではg0レジスタのタグを調べ、もしもタグが変数であれば次のjump命令へ、それ以外であればfail命令に分岐する。

インライン展開を行わないコードを実行した場合で、組込述語var/1を基本命令で記述した場合は１２命令が実行されるが、インライン展開を行ったコードでは７命令が実行される。更に前者の実行にはallocate、call_built_inというコストの高い命令が実行されるので、実行時間では両者の差は２倍以上となる。

３．２．算術演算述語の展開

組込述語is/2の右辺は、基本的な四則演算組込述語を用いてPrologレベルで展開し、更にこれらをインライン展開する。

例えば、節

```
f(X,Y,Z,U) :- U is (X + Y) * Z.
```

はまず次のように変換する。

```
f(X,Y,Z,U) :-
        add(X,Y,Temp1),
        mul(Temp1,Z,Temp2),
        U = Temp2.
```

次にadd/3、mul/3、=/2をインライン展開して次のようなコードを生成する。

```
        deref       A0,A0
        deref       A1,A1
        deref       A2,A2
        add         g0,A0,A1  % g0 := A0 + A1
        mul         g0,g0,A2  % g0 := g0 * A2
        unify       A3,g0
```

複雑な式に対するレジスタの割当て技法は、通常の言語（例えばFORTRAN）のレジスタ割当て技法がそのまま使用できる。

４．遅延分岐命令を用いたappendプログラムのコンパイル例

Prologのプログラムでは、リストの各々の要素に対して処理を行ないながらリストをニルになるまで辿るプログラムが非常に多い。小型化版ＣＨＩでは、このような繰り返しを高速に実行する為にリストに対するインデクシング命令ならびに遅延インデクシング命令を導入した。この命令を用いてappend/3をコンパイルすると次のようなコードを生成する。

```
        switch_on_list          $1,$3,$0
$0:     try_me_else             $2
        get_nil                 A0
$1:     get_register_value      A1,A2
        proceed
$2:     trust_me_else_fail
        get_list                A0
$3:     unify_register_variable A3
        unify_register_variable A0
        get_list                A2
        unify_resiger_value     A3
        delayed_switch_on_list  $1,$3,$0
        unify_register_variable A2
```

[遅延]インデクシング命令[delayed_]switch_on_listは、A0レジスタの内容がニル、リスト、変数であった場合に３方向に分岐し、それ以外の場合には失敗する。更にA0レジスタの内容がリストであった場合には、unify命令で使用するSPポインタがリストの第１要素を指し示すようにする。この命令を用いることによって、A0レジスタの内容がニルであるかどうかを判定するget_nil命令ならびに指定されたレジスタの内容がリストであるかどうかを判定するget_list命令の実行が省略でき、execute命令も不要となる。更に遅延分岐による効果として分岐先の命令取り出しの待ち時間がなくなることによって約400KLipsの実行予測性能が得られる。

５．終りに

ＳＵＰＬＯＧコンパイラにおける最適化技法について述べた。レジスタ操作を中心とした低レベルの命令セットを用いることによってきめ細かな最適化が可能であること、また専用機においてはリストに対する遅延インデクシング命令のような特殊な命令を用意してコンパイルすることによってリストを高速に辿ることが可能であることがわかった。今後はこれらの最適化技法を実用規模の応用プログラムに対して適用し評価を行う予定である。

参考文献

[1] Nakazaki,R. etal: Design of a Co-operative High Performance Sequential Inference Machine (CHI), NEC Research & Development, no.80.1986

[2] 中嶋　他: 逐次型推論マシンCHI小型化版の設計思想, 情報処理学会第33回全国大会論文集, 1986

[3] Warren.D.H.: An Abstract Prolog Instruction Set, TR309, AI Center, SRI International,1983

[4] 小長谷　他: 高性能PROLOGマシン(CHI)における最適化技法, 情報処理学会第32回全国大会論文集, 1986

[5] 小長谷　他: 大規模論理型言語プログラムにおけるコンパイラ最適化の効果, 1986, to appear