4B-10

# ＰｒｏｌｏｇマシンＰＥＫにおける ｐｒｏｌｏｇ中間コードについて

宮本昌也* 和田耕一** 金田悠紀夫* 前川禎男*
* 神戸大学工学部システム工学科 ** 神戸大学大学院自然科学研究科

## １．はじめに

逐次実行型ｐｒｏｌｏｇマシンＰＥＫ[1]は、ストラクチャシェアリング方式を基本に、Ｐｒｏｌｏｇプログラムの高速実行を目的に製作されたマシンである。その目的達成のためコンパイラには、ＰＥＫのハードウェアを有効に利用しえる中間コードが必要である。

今回、設計した中間コードは、Ｄ．ＷａｒｒｅｎのＡｂｓｔｒａｃｔ Ｐｒｏｌｏｇ Ｉｎｓｔｒｕｃｔｉｏｎ Ｓｅｔ[2]（以下、ＡＰＩＳと略）を参考にしている。引数のレジスタ割り付けの基本方針はＡＰＩＳに従った。ストラクチャの処理はコピー方式ではなく、シェアリング方式を採用した。この中間コードの採用により、決定的ａｐｐｅｎｄプログラムで４０７．９ＫＬＩＰＳの速度を得ることができた。

## ２．基本方針

今回の中間コード設計の基本方針は以下の通りである。

① 引数のレジスタ割り付け。

ＰＥＫは、２ポート入力、１ポート出力のレジスタファイル持っている。このレジスタファイルに、クローズの引数内容を割り付けることにした。

レジスタファイルへの書込みは、他のスタック、メモリへの書き込みと同時に行うことができる。このため、引数のレジスタ割付のインプリメントを加えても、マイクロ命令ステップは増加しない。クローズの引数に対するデリファレンスが減り、大幅にマイクロ命令ステップを削減できた。

現在、インタプリタとの競合があるため、４引数まで可能となっている。

② モード宣言

各クローズの引数は入力、出力が決まった形で使われることが多い。モード宣言とインデキシングを共用した場合、次に来るｃａｌｌｅｒ引数のタイプを絞り込むことが可能となり、分岐命令を減らすことができる。

入力用に宣言された引数では、ｃａｌｌｅｒ引数のデリファレンスの必要がないとする。多くの場合リテラルとストラクチャが来ることになる。第１引数の場合は、インデキシングによってどちらか一方に決まる。このためタイプチェックの必要が無くなってしまう。

出力用に宣言された引数では、ｃａｌｌｅｒ引数は未定義であると考え、その変数セルへの代入動作だけを行う。

③ 成功時情報の格納の最適化

ｃａｌｌについては、ｆｉｒｓｔ＿ｃａｌｌ，ｍｉｄ＿ｃａｌｌ，ｌａｓｔ＿ｃａｌｌの３種類とした。それぞれ最初のボディ、次ぎ以降のボディ、最後のボディに対応する。これにより必要な時にのみ成功時情報を格納する事にした。

④ グーロバル変数のみとする

ローカル変数を考えないことにより、プロセスメモリ上に変数環境を取らなくてよい。これによりプロセスメモリベースレジスタのアクセスを減らすことができる。また、ＴＲＯの対象となる情報が、成功時情報のみになる。この処理は前述したｃａｌｌ命令が行い、ＴＲＯは行わない。

⑤ ストラクチャシェアリング方式

ＰＥＫはストラクチャシェアリング方式を基に設計されており、データバスはフレーム部を持っている。

ストラクチャ引数のユニフィケーション命令はＡＰＩＳと大きく異なっている。ｃａｌｌｅｒの引数は共有メモリから読みだされる。

## ３．命令セット

各中間コードの分類は次の通りである。

① 実行制御命令

| | | |
|---|---|---|
| switch_on_term | try_me_else | first_call |
| neck | retry_me_else | mid_call |
| proceed | trust_me_else_fail | last_call |
| | cut | execute |

② ｐｕｔ命令

put_frame　put_var　put_val　put_atom
put_skel　put_shadow

put_frameはボディ部のグローバルスタックベースをセットする。各ｐｕｔ命令列の先頭に置かれる。

③ ユニフィケーション命令

ここではモード宣言時の命令セットを示す。命令は入力、出力、入出力の３タイプに分かれる。また、それぞれレベル０、レベル１に分かれる。レベル０の命令は第１引数か否かによって２つに分類される。第１引数ではswitch_on_term命令が前処理を先に行うためほかの命令よりコンパクトになる。レベル１命令は、ストラクチャの最終引数か否かで分類される。最終引数の場合レベル０のユニフィケーションのための準備を行う。

入力用

uin_gvar0　uin_atom0　uin_skel0　uin_gref0
uin_gvar0_1 uin_atom0_1 uin_skel0_1
uin_gvar1　uin_atom1　uin_skel1　uin_gref1
uin_gvar1_1 uin_atom1_1 uin_skel1_1 uin_gref1_1

---

PLM code on the Prolog Machine PEK
MASAYA MIYAMOTO, WADA KOUICHI, YUKIO KANEDA, SADAO MAEKAWA
KOBE University.

出力用

| | | |
|---|---|---|
| uout_gvar0 | uout_atom0 | uout_skel0 |
| uout_gvar0_1 | uout_atom0_1 | uout_skel0_1 |
| uout_gref0 | | |

レベル１の出力用命令は必要ない。uout_skel0,uout_skel0_1命令は、自らのスケルトンをcaller側変数セルに代入するだけである。

入出力用

| | | | |
|---|---|---|---|
| ugvar0 | uatom0 | uskel0 | ugref0 |
| ugvar0_1 | uatom0_1 | uskel0_1 | ugref0_1 |
| ugvar1 | uatom1 | uskel1 | ugref1 |
| ugvar1_1 | ugvar1_1 | ugvar1_1 | ugref1_1 |

## ４.中間コードの例

図１にレベル１の入力用変数のマイクロ命令を示す。グローバルスタックへの代入は２ステップで行われる。ステップ１では、変数番号とグローバルスタックベースとから変数セルのアドレスがｆｔ２にセットされる。ステップ２では、コモンメモリからの内容が引数レジスタとグローバルスタックとの両方に書き込まれる。これでcallee側変数への書込みが完了する。ステップ３ではオフセットのインクリメントが行われる。このように、ステップ２でレジスタ割り付けを行ってもマイクロ命令は増加しない。

図２はlast_callのマイクロ命令である。ステップ２でプロセスメモリからレジスタへの書込みと、次の述語へのジャンプを同時に行っている。ＰＥＫではジャンプ命令は他の処理と平行して行うことができる。

## ５.評価

図３に決定的ａｐｐｅｎｄプログラムとその中間コードを示す。最適化は既に行われている。このコードの命令数と実行時間を表１に示す。サイクルタイムは１２０nsecと１６０nsecである。モード宣言のインプリメントにより、第２クローズの１回のユニフィケーションが、９マイクロステップとコンパクトになった。シミュレータを用いて計測した結果、要素３０の決定的ａｐｐｅｎｄでは４０７.９ＫＬＩＰＳとなった。また、ａｐｐｅｎｄプログラムを使った要素３０のｒｅｖｅｒｓｅの実行速度は３４０.７ＫＬＩＰＳである。

表１．Ａｐｐｅｎｄプログラムの評価

| 命　令 | 命　令　数 | | 実行時間 [nsec] | |
|---|---|---|---|---|
| | [1,..,30]-[30] | [] | [1,..30]-[30] | [] |
| switch_on_term | 3*30=90 | 3 | 440*30=13200 | 440 |
| uin_atom0_1 | | 3 | | 440 |
| uout_gref0 | | 2 | | 240 |
| proceed | | 3 | | 480 |
| uin_skel0_1 | 1*30=30 | | 160*30=4800 | |
| uin_gvar1 | 3*30=90 | | 480*30=14400 | |
| uin_gvar1_1 | 3*30=90 | | 480*30=14400 | |
| uout_skel0 | 2*30=60 | | 280*30=8400 | |
| neck | 1*30=30 | | 160*30=4800 | |
| put_frame | 1*30=30 | | 160*30=4800 | |
| put_var | 1*30=30 | | 160*30=4800 | |
| execute | 1*30=30 | | 160*30=4800 | |
| ｔｏｔａｌ | 480 + 11 = 491 | | 74400+1600=76000 | |

【参考文献】

１.田村直之,和田耕一,小畑正貴,金田悠紀夫,前川禎男,日根俊治：シーケンシャル実行型Ｐｒｏｌｏｇマシンｐｅｋ、情報処理学会論文誌、Ｖｏｌ.２６ Ｎｏ.５ Ｓｅｐ.１９８５

２.Ｗａｒｒｅｎ,Ｄ.Ｈ.Ｄ：Ａｎ Ａｂｓｔｒｕｃｔ Ｉｎｓｔｒｕｃｔｉｏｎ Ｓｅｔ, ＳＲＴ Ｔｅｃｈｎｉｃａｌ Ｎｏｔｅ ３０９（１９８３）

３.Ｗａｒｒｅｎ,Ｄ.Ｈ.Ｄ：Ｉｍｐｌｅｍｅｎｔｉｎｇ Ｐｒｏｌｏｇ, Ｄ.Ａ.Ｉ Ｒｅｓｅａｒｃｈ Ｒｅｐｏｒｔ Ｎｏ.３３

```
uin_gvar1(GVAR, SHD), [

ft2 = alu( xrb(qq) = ra(gs_bottom);#GVAR )
gs(ft2) = alu( xrb(SHD) = rd2 ), push(ts)
alu( value(xrb(wk0)) = xrb(wk0) + 1 )
```

図１．uin_gvar1のマイクロコード

```
last_call(PRED), [
gensym(NEXT),

alu( rb(s_pc) = pm(pbr_s_pc) )
alu( rb(s_base) = pm(pbr_s_base) ), jmp(#PRED)
```

図２．last_callのマイクロコード

```
-:mode(app(+,+,-)).

app([],Y,Y).
app([H|X],Y,[H|Z]):-app(X,Y,Z).


     org('$0800').

x_0001: try_me_else(x_0003).
x_0002: uin_atom0_1('$30000').
        uout_gref0(arg3, arg2).
        proceed('$00001').

x_0003: trust_me_else_fail.
x_0004: uin_skel0_1('$C7500').
          uin_gvar1('$80000',wk3).
          uin_gvar1_1('$80001',arg1).
        uout_skel0('$C7503',arg3).
        neck('$00004').
        put_frame.
        put_var('$80003',arg3).
        execute(app).

        backtrack.

app:
        switch_on_term(i_fail, x_0001, x_0002,
                               x_0004, i_fail).

        org('$0080').

        jump(app).
```

図３．Ａｐｐｅｎｄプログラムとその中間コード展開