# ＰＳＩ－ＩＩ　の性能評価

## 3Y-5　— Ｉｆ＿Ｔｈｅｎ＿Ｅｌｓｅ，Ｎｅｃｋ＿Ｃｕｔ —

立野　裕和* 近藤　誠一* 中島　浩* 池田　守宏*　中島　克人**

＊：三菱電機，　＊＊：ＩＣＯＴ

### １．はじめに

第五世代プロジェクトの一環として、マルチＰＳＩの要素プロセッサ及びフロント・エンド・プロセッサとして用いられるＰＳＩ－ＩＩを開発した。ＰＳＩ－ＩＩはＷＡＭ[1]をベースとしたＫＬ０処理系が実装されている。本稿では、if_then_else最適化[2]、neck_cut最適化[3]に関する性能評価結果について報告する。

### ２．ｉｆ＿ｔｈｅｎ＿ｅｌｓｅ最適化

if_then_else最適化はＰＳＩで導入された最適化技法であり、図１に示すような従来のクローズ・インデキシングが不可能な述語の高速実行のために用いられる。

```
p(X,Y,Z):- integer(X),X>Y,!,q(X,Z)
p(X,Y,Z):- r(X,Z).
```

図１　if_then_else最適化の対象となる述語例

ＰＳＩの場合、if_then_else最適化はＯＳの高速化が主な目的であったことなどからＯＳ開発者に特別のシンタックスを提示しプログラム・ソースを書き直すことで導入した。しかしＰＳＩ－ＩＩではコンパイラの最適化手法のひとつとして採用しており表１の条件を満たすクローズに対し、if_then_elseのコードが生成される。

表１　if_then_else最適化の適用条件

(1) オルタネイト・クローズが１つ以上存在する。
(2) ヘッド引数がすべて異なる変数である。
(3) ボディにカット命令があり、そのカットの前は呼び出し側の未定義変数に値を束縛したり、不正入力などのエクセプションが発生しない組込述語。

ＰＳＩ－ＩＩではクローズ内ＯＲを図２に示すように展開しており、この条件は展開後のクローズに対して適用される。そのため、クローズ内ＯＲにおいてはＯＲの左の枝が条件(3)を満たすだけでよい。

```
p(X,Y,X):-(X>Y, !, q(X,Y) ; r(X,Y)),s(Y).
        ↓
p(X,Y,X):- $or_p(X,Y),s(Y).
$or_p(X,Y):- X>Y, !, q(X,Y).
             if部    then部
$or_p(X,Y):- r(X,Y).
             else部
```

図２　クローズ内ＯＲの展開

```
        jump_unless_less_than X2 X1 Else
        execute  q/2
Else:
        execute  r/2
```

図３　図２のＯＲ部分のコンパイル結果

図３に図２のＯＲ部分のコンパイル結果を示す。if部の組込述語はjump_unless_less_than命令にコンパイルされる。この命令では、比較が失敗すると指定されたアドレスに単に分岐する。このような命令を導入することにより以下に示す処理を省略し高速なクローズ選択を実現している。

(1) backtrackフレームを生成しない。
(2) (1)によりfail処理を省略。

### ３．ｎｅｃｋ＿ｃｕｔ最適化

表２に示す条件を満たす述語に対してはneck_cut最適化がなされる。表１よりも緩い条件となっているが、適用時の速度向上もif_then_else最適化よりはいくぶん低くなる。

表２　neck_cut最適化の適用条件

(1) 最後のクローズを除く全てのボディー部にカットがあり、カットのまえにユーザー定義述語の呼出しが無いこと。
(2) 各クローズのカット命令以前のユニフィケーション処理において引数レジスタを破壊することなくコンパイルが可能なこと。

図４にneck_cut最適化が適用可能な述語例を示す。このような述語では、swich_on_term命令によるクローズ選択が行なえないため、通常はtry,retry,trust等の命令で逐次に実行される。ところが、最後を除くすべてのクローズはカットを含んでおり、また、カットの前はすべて組込述語であるので、これは決定的な述語といえる。従って、以下の処理方式をとることにより高速化を図るのがneck_cut最適化である。

(1) backtrackフレームをスタック上に生成しない。
(2) カットの実行まで必要最低限のbacktrack情報を専用のレジスタ上に保持する。（ただしＰＳＩ－ＩＩでは変数のトレール情報は通常のトレール・スタック上で行なっている。）

---

Performance Evaluation of PSI-II -- If then else &Neck cut optimizeation --

H.Tateno *, S.Kondoh *, H.Nakashima *, M.Ikeda *, K.Nakajima **

*:Mitubishi Electric Co.,Ltd, **: ICOT

backtrack情報を専用のレジスタに保持した状態をfast_modeと称しており、このモードへの切り換え、次クローズ選択、fast_modeの終了のために、それぞれ、fast_try,fast_retry,fast_trust命令などを導入した。また、cut_me命令ではfast_modeであるかどうかによってbacktrack情報のキャンセルのしかたを切り換えるように仕様変更した。

```
p(X,X,Z):- !,q(X,Z).
p(X,Y,Z):- integer(Y), 10 = X + Y, !, s(Z).
p(X,Y,Z):- t(X,Y,Z).
```

図４　neck_cut最適化の対象となる述語例

図５に図４のコンパイル・コードを示す。

```
      fast_try          Next/3
      get_variable_t    X04  A01
      get_value_t       X04  A02
      cut_me
      put_value_t       X03  A02
      execute           q/2
Next:
      fast_retry        Last
      integer           A02
      add               A01  X02  X04
      get_integer       10   X03
      cut_me
      put_value_t       A03  A01
      execute           s/1
Last:
      fast_trust
      execute           t/3
```

図５　図４のコンパイル・コード

## ４．性能評価

表３にクイック・ソートの実行性能比を示す。プログラムはif_then_else、neck_cut最適化の効果を比較するためPrologコンテストでのベンチ・マーク・プログラムのpartitionの第２,第３クローズをクローズ内ＯＲに書き換えたものも使用した。以下にそれらを示す。

qsort0: Prologコンテストでのベンチ・マーク
（無修正）

```
patition([X2|List],X1,[X2|List1],L):-
 X2<X1, !, partition(List,X1,List1,L).
partition([X2|List],X1,L1,[X2|List2]):-
 partition(List,X1,L1,List2).
```

qsort1: 通常のクローズ内ＯＲに変更。

```
partition([X2|List],X1,L1,L2):-
 (X2<X1,!, L1 = [X2|List1],
  partition(List,X1,List1,L2);
  L2 = [X2|List2],partition(List,X1,L1,List2))
```

qsort2: クローズ内ＯＲをＰＳＩで導入した特別なシンタックスに変更。

```
partition([X2|List],X1,L1,L2):-
 (X2<X1 -> L1 = [X2|List1],
  partition(List,X1,List1,L2);
  L2 = [X2|List2],partition(List,X1,L1,List2))
```

プログラムの実行性能は条件節が全て成功した場合と全て失敗した場合の平均値とした。また実行性能比は、qsort0に対しneck_cut最適化を適用せずにコンパイルしたコードの実行性能を基準としている。

表３　クイック・ソートの実行性能比

| プログラム | 実行性能比（適用した最適化） |
|---|---|
| ｑｓｏｒｔ０ | ０．７８　（neck_cut最適化） |
| ｑｓｏｒｔ１ | ０．４９（if_then_else最適化） |
| ｑｓｏｒｔ２ | ０．４５（if_then_else最適化） |

実用的なプログラムの一例としてコンパイラ（Ｋｃｏｍｐ）にif_then_else、neck_cut最適化を適用した場合の実行性能を表４に示す。Ｋｃｏｍｐに対してプログラムの書き換えはおこなっていない。

表４　Ｋｃｏｍｐの実行性能

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 実行速度(msec) | 106.67 | 96.64 | 93.70 | 88.73 |
| 性能比 | 1.00 | .906 | .878 | .832 |

1) non optimize　2) only neck_cut
3) only if_then_else 4) neck_cut & if_then_else

if_then_else,neck_cut最適化の導入によりコンパイラのような応用的なプログラムにおいてその実行性能が約１５％程度向上することが確認された。最適化を行わない場合、マイクロプログラムの全実行ステップの約２７％をtry系及びbacktrack処理が占めていた[4]ことを考えると今回導入した最適化によりそれらの処理の半分以上の処理が省略されたと言える。

## ５．まとめ

if_then_else、neck_cut最適化の導入により、従来のクローズ・インデキシングでは対象外であった決定的なクローズにおいても、backtrackフレームを生成する事なくクローズ選択が可能になること、また、この最適化により、Ｋｃｏｍｐ（コンパイラ）のような実用的な応用プログラムにおいて約１５％の高速化が実現できる事を示した。最後に本研究にあたり貴重な助言をいただいたＩＣＯＴ及び関連メーカーの方々に深く感謝します。

参考文献

(1) D.H.D.Warren,.
An Abstruct Prolog Instruction Set.
Tech Note 309,AI Center,SRI 1983

(2) 高木他 "KLO機械語へのIF_THEN_ELSE 制御構造の導入によるプログラム実行効率の向上について"
ICOT TM-0104 1985

(3) 中島 浩他 "マルチPSI要素プロセッサPSI-IIの最適化手法"
第３３回情報処理全国大会 7B-4

(4) 中島 克人他 "PSI-IIの性能評価(1)
第３５回情報処理全国大会 6B-7