ＰＳＩ−Ⅱ の性能評価（２）

−− 機械命令評価 −−

6B−8

稲村 雄　　近藤 誠一　　中島 克人
（ＩＣＯＴ）　　（三菱電機）　　（ＩＣＯＴ）

１．はじめに

ＰＳＩ−Ⅱでは１６ＫＷの大容量ＷＣＳと１０２４種の命令コード数を背景にして，ＷＡＭ基本命令セット[1]に多くの最適化命令を追加することで性能向上を図っている．本稿ではこれらの最適化命令の導入による性能向上の度合を測定し，その有効性を検証する．

２．最適化方針

２．１．クローズインデクシングの細分化

データタイプによるインデクシングの為に汎用的な Switch_on_term 命令に加えて，キーとなる引数のタイプが一種類（アトミック or リスト or 構造体）だけのクローズから成る述語のために Jump_on_non_constant 等の特定データタイプ以外のタイプが現われた時に分岐させる命令を用意した．これは同じ引数位置で同一のデータタイプを持つ様なプログラミングスタイルが多いと思われるためであり，又，ＰＳＩ−Ⅱの機械語では Jump_on_non_XX 命令を２語命令で実現できる（Switch_on_term 命令は３語命令）ため，これらの命令の導入によりコード量も削減される

２．２．複数命令の統合

コンパイル結果のコード中に，ある同じパターンの連続した命令列が頻繁に現われることがある．これらの命令列を統合して一つの新しい命令に置き換えることで以下のような効果が期待できる．

①コード量の削減
②マイクロステップ数の減少
③命令フェッチ回数の減少によるメモリアクセス回数の削減

現在ＰＳＩ−Ⅱに導入されている統合化命令数は２０種類程度である．これらのうち主な命令について以下に述べる．

・Get_list 命令 + 変数 Unify 命令

Car部,Cdr部ともに変数であるようなリストがヘッドにある場合に Get_list 命令に続けて変数をユニファイする命令が２つ発行される．この２つのユニファイ命令の組み合わせは，変数に初出か否か，Parmanent 変数か否か，の４種類があるため，１６通りとなる．現時点ではこの中で出現頻度が高いと思われる５個の命令を用意している．

・Execute 命令 or Call 命令 + インデクシング命令

Execute 命令，又は Call 命令の呼び先に Switch_on_term 等のデータタイプによるインデクシング命令がある場合の為に Execute 命令（Call 命令）と Switch_on_term 命令をマージした命令 Execute_and_switch（Call_and_switch）を用意した．

但し，ゴール呼出し側にインデクシング機能を含めるため，飛び先ラベルが１個から４個へと増加し，Execute(Call)_and_switch 命令は３語命令となる(Call 命令，Execute 命令は１語命令)．従って，これらの命令の導入により呼出し側のコード量は増加する．

・Execute 命令 + Deallocate 命令

Execute 命令と環境解放命令である Deallocate 命令をマージした命令 Execute_with_deallocate を導入した．上記同様インデクシングまで同時に行うExecute_with_deallocate_and_switch 命令も用意してある．

２．３．コンパイル結果例

以上の最適化命令の幾つかは，その有効性を評価するために，その命令を使用するか否かをコンパイル時に指定できる様になっている（コンパイル時に設定するOptimize Levelの変更による）．それらの命令を使用した場合と使用しなかった場合でのコンパイル結果の違いを図１に示す．使用したプログラムは Append である．

```
                append([],L,L):-!.
                append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

append/3:  try_me_else L1          append/3:  jump_on_non_list A0, Lnil,
           get_nil A0                                          Lelse, Lvar
           get_value_t A2, A1          Llist: get_vart_vart_list A0,A3,A0
           cut_me_and_proceed                 get_valt_vart_list A3,A3,A2
      L1:  trust_me_else_fail                 execute_and_switch A0,Lconst,
           get_list A0                                         Lstruct,
           unify_variable_t A1                                 Lvar,Llist
           unify_variable_t A0    Lnil: Lconst:  get_nil A0
           get_list A2                        get_value_t A2, A1
           unify_value_t A3                   cut_me_and_proceed
           unify_variable_t A2          Lvar:  try Lnil
           execute append/3                    try Llist
                                     Lstruct:  fail

  (i)  No Optimize                          (ii) Full Optimize
```

図１ コンパイル結果例

３．最適化命令評価

３．１．コード量の増減

図２に幾つかのプログラムに関して Optimize Level を変えたコンパイルで得られたコード容量の増減の模様を示す．一般に，インデクシング命令の導入はコード量の増加をもたらし（変数で呼ばれたときのために try 命令列は減らせない上に，予期しないタイプのためにも飛び先を用意する必要がある），複合命令はコード量を減少させる（ゴール呼出し命令とインデクシング命令とをマージした場合を除く，理由は前述の通り）．

Program	(i)	(ii)	(iii)	(iv)
Append	12(8)	15(11)	17(13)	
Qsort	48(40)	56(48)	62(54)	66(58)
Harmonizer	3406(3380)	3964(3967)	4023(3997)	4055(4029)

コンパイル条件 〜
(i) : No optimize
(ii) : Clause Indexing
(iii) : (ii) + Execute(_with_deallocate)_and_switch
(iv) : (iii) + Call_and_switch
括弧内 : (Get_list + 変数 Unify) 命令使用時

図 ２ コンパイル条件によるコード量の増減（単位：ワード）

Performance Evaluation of PSI-Ⅱ (2) -- Machine Instruction Evaluation --
Y.Inamura*1, S.Kondoh*2, K.Nakajima*1
*1:ICOT, *2:Mitsubishi Electric Co. Ltd.

3．2．最適化命令の出現頻度

2節で述べた最適化命令の内，ゴール呼出し命令＋インデクシング命令と Get_list＋変数 Unify 命令の2種類について，その出現頻度を以下のような方法で測定した。結果を図3，図4に示す。

①Call_and_switch 命令

Call_and_switch 命令を出さないようにコンパイルしたプログラムを実行，実行中に Call 命令の次命令が何であったかを記録し，全 Call 命令中で次命令がインデクシング命令である(Call_and_switch 命令に置換され得る)ものの割合を求める（Execute_and_switch 命令，Execute_with_deallocate_and_switch 命令についても同様），測定の対象としたプログラムには実用的かつ比較的大規模なものとして，Compiler を選んだ。それぞれのプログラムの内容については後述する。

この測定結果から，ゴール呼出し命令の呼び先では，リスト引数を待つ場合に発行される Jump_on_non_list 命令が現われるケースが多い事が分る。Call/Execute_and_switch 命令はこのケースを優先させる様に実現されている。

	T	S	C	V	L	U	N
CALL	47.8 %	7.4 %	1.9 %	19.3 %	7.7 %	1.4 %	14.5 %
EXEC	56.2 %	3.6 %	3.7 %	8.5 %	13.7 %	1.3 %	13.0 %
EXECD	50.2 %	1.6 %	0.5 %	0.5 %	28.7 %	18.5 %	13.0 %

T : Try_me_else
S : Switch_on_term
C : Jump_on_non_constant
V : Jump_on_non_vector
L : Jump_on_non_list
U : Jump_on_non_unifiable_value
N : 非 Indexing 命令

図 3 ゴール呼出命令に続く Indexing 命令 出現頻度

②Get_list 命令＋変数 Unify 命令

Get_list 命令と変数 Unify 命令をマージした命令（以下 Get_X_Y_list 命令と記述）なしでコンパイル，実行させ，Get_list 命令に続く2つの 変数 Unify 命令の出現頻度をパターン別に集計し全 Get_list 命令に対するそれぞれのパターンの出現割合を求める。これによってGet_X_Y_list 命令の中でどの組み合わせが重要かを調べ，新しい命令の追加，もしくは現存の命令の廃止を決定するための指針とする。測定に使用したプログラムは①と同様 Compiler である。

変数 Unify 命令の出現頻度が Get_list 命令全体に占める割合は，約３０％とかなり低いことが分かった。中では，二つとも初出の Temporary 変数を待つ場合が予想通り最多であった。又，既出の変数を待つケースも予想以上に存在し，再考を促される結果となった。

RT,RT	RT,LT	LT,LT	LT,RT	LT,RP	RP,RP	else	N.B.
15.9 %	4.4 %	2.8 %	1.8 %	1.1 %	1.1 %	0.4 %	71.5 %

RT : 初出 Temporary 変数
RP : 初出 Permanent 変数
LT : 既出 Temporary 変数
LP : 既出 Permanent 変数
N.B. : 非変数 (unify_atom etc)

図 4 Get_list 命令に続く Unify 命令列 出現頻度

3．3．実行速度の向上

幾つかのベンチマークプログラム並びに実用プログラムについて，Optimize Level を８段階に変えてコンパイルしたものを実行し，それぞれの実行速度を測定した。これによって各最適化命令の導入による速度性能の増加が段階的に確認できる。実行したプログラムは以下の通り。

①ベンチマークプログラム

- Append （１０００要素）
- Quick_Sort （５０要素）
- 8-Queens （全解）
- Harmonizer ～ 入力されたメロディーにコードをつけるプログラム。今回は" 赤トンボ" のメロディーを使用。

②実用プログラム

- コンパイラ ～ ＰＳＩ－Ⅱ上のＥＳＰコンパイラ。コンパイラ自身がＥＳＰで記述されているため今回ベンチマークとして用いた。測定はこのコンパイラによる Append 及び Qsort のプログラムのコンパイルに要する時間を計ることで行われた。

結果を図5に示す。

Program 名	(1)	(2)	(3)	(4)
Append	1.00	2.93	3.24	3.24
(L)	1.03	3.45	3.89	3.89
Qsort	1.00	1.28	1.32	1.34
(L)	1.01	1.32	1.38	1.38
Nqueens	1.00	1.03	1.03	1.03
(L)	1.01	1.04	1.04	1.04
Harmonizer	1.00	1.48	1.45	1.48
(L)	0.99	1.47	1.43	1.45
Compiler	1.00	1.19	1.98	1.99
(L)	1.01	1.19	1.99	2.00

単位 : No Optimize, Get_X_Y_list 非使用時を基準，相対値で表示

計測条件
(1) : No Optimize
(2) : Clause Indexing
(3) : (2) + Execute(_with_deallocate)_and_switch
(4) : (3) + Call_and_switch
(L) : Get_X_Y_list 使用時

図 5 Optimize Level による速度性能の向上

4．結論

Append 等の決定的に実行されるプログラムに対しては，クローズインデクシングの導入により，環境フレーム作成を可能な限り避ける事が，性能改善に大きな効果を持つことが確認された。又，この様なごく小規模なプログラムに対しては，本稿で述べた細かな最適化が有効である事が分かった。

更に，Harmonizer，Compiler 等の比較的大規模なプログラムに対しても相応な効果が確認できた。

反面，Nqueens 等の非決定的に実行されるプログラムに対しては大きな効果はみられず，これらに対する最適化は今後に残された課題である。

5．おわりに

今後は，他の多くのプログラムに対して計測を行いこれらの命令の有効性を評価するとともに，更なる最適化命令の導入を検討して行く予定である。

本研究において，多くの貴重な意見をいただいたＩＣＯＴ並びに関連メーカーの方々に深く感謝致します。

参考文献

[1]:D.H.D. Warren, An Abstruct Prolog Instruction Set TR309, SRI, 1983.