7B-4

マルチPSI要素プロセッサ PSI－IIの最適化手法

中島　浩．近山　隆．中島　克人
（三菱電機）　（新世代コンピュータ技術開発機構）

1．概要

我々は，逐次型の推論マシンを格子状に結合したマルチプロセッサである，マルチPSIを開発している。マルチPSIの要素プロセッサであるPSI－IIは，論理型言語の逐次実行に適したアーキテクチャを採用している。ここでは，PSI－IIにおけるKL0 の処理系のついて述べる。KL0 処理系は，D.H.D.Warrenが文献[1] で提案した方式を基本にしているが，高速化や，KL0 特有の諸機能を実現するために，様々な最適化や拡張が施されている。以下，その主なものについて述べる。

2．組込述語

組込述語は，文献[2] に示されるように，非常に頻繁に呼び出される。従って，その実行速度（特に引数の受渡しの速度）はシステムの性能を大きく左右する。

PSI－IIにおける，組込述語への引数受渡しの方式は，一般的に次のように表される。

```
{ put input arguments }
call builtin predicate
{ get output arguments }
```

なお，入力引数の"put" は通常の述語呼出しと同様の命令列（即ち引数レジスタへの書込み）を用いる。また，出力引数の"get" は，ヘッド引数のユニフィケーションに用いる命令列で実現される。

さて，組込述語の呼出しにおける，引数レジスタの割付けの方法として，次の２つが考えられる。

①（通常の述語呼出しと同様に）引数レジスタを順番に用いる。

②任意の引数レジスタ（一時変数レジスタ）を用いる。

方法①は組込述語の処理において，引数レジスタへのアクセスが（比較的）容易に行えるというメリットがある。これに対し，方法②は一時変数レジスタの割付けの自由度が増し，レジスタ間転送の削除などによる高速化が期待できるというメリットがある。

これらの得失を検討した結果，我々は次のような理由から方法②を採用した。

(a) 組込述語の呼出し命令を，"get" ，"put" 等の命令と同じ形式にすることにより，引数レジスタのアクセスをサポートするハードウェア機構を使用した，高速アクセスが可能となる。

(b) 一時変数レジスタの割付けの最適化の効果は，かなり大きいことが予想される。

最適化の効果を，図１に示す述語"nth_elem"の第２クローズを用いて示す。図２(a) は方法①に，(b) は方法②にしたがった命令列である。両者の実行速度を，マイクロ命令の実行ステップで見積もると，それぞれ１７stepと１３stepであり，約１：１．５の性能差があることがわかる。なお，加減算のオペランドの一方が定数であることが多いことを予想し，レジスタに対する即値の加減算を行う組込述語命令，"add_constant"および"subtract_constant" を用意している。"subtract_constant" を用いると，上記の例での実行ステップは，１１stepに改善される。

```
nth_elem(0,[E|_],E) :-!.
nth_elem(N,[_|L],E) :-
        N1 is N - 1,          % subtract(N,1,N1)
        nth_elem(N1,L,E).
```

図１

(a)	(b)
get_list A1	get_list A1
unify_void 1	unify_void 1
unify_variable X3	unify_variable A1
get_variable X4,A2	
put_integer 1,A1	put_integer 1,X3
subtract	subtract A0,X3,A0
get_variable A0,A2	
put_value X3,A1	
put_value X4,A2	
execute nth_elem/3	execute nth_elem/3

図２

3．カット

多くの実用的プログラムでは，高速化，メモリ消費量の削減，プログラムの見通しを良くする，などの目的でカットが多用される。従って，カット処理の高速化も重要な要素である。PSI－IIでのカット処理のポイントとして，以下の３点があげられる。

①（結果的に）不要なカットをすばやく認識する。

②除去すべき選択肢をすばやく発見する。

③できるかぎりスタックを縮める。

以下それぞれについて述べる。

(1) 不要なカット：　クローズ・インデクシングの結果，カットが無効化されることが少なくない。図３に示すような述語では，その第一引数が変数でなく，クローズ・インデクシングが完全に成されていれば，カットは不要

Optimization Method in Multi-PSI Element Processor (PSI-II)

H.Nakashima(*),T.Chikayama(**),K.Nakajima(**)
*:Mitsubishi Electoric Corp. **:Inst. for New Generation Computer Tech.

```
p(a,...):-!,...
p(b,...):-!,...
p(c,...):-!,...
p(d,...):-!,...
```
図3

```
p:-q,!,...
```
図4

である。しかし，クローズ・インデクシングのふるまいをプログラマが完全に認識していることを期待することはできないので，このような状況が頻繁に起こると考えなければならない。そこで，以下のような方法を採用した。

(a) 実行中のクローズが決定的に選択されたかどうかを示すフラグ"DET" を設ける。
(b) "DET" は述語呼出し時にゼットされ，"try_me_else" や"try" などでリセットされる。
(c) すべての通常の述語呼出しに先立つカット処理では，"DET" を判定し，決定的であればなにもしない。

この方法により，最小限の手間で，不要なカットを認識することができる。なお，この手法は通常の述語呼出しに先立つカットにのみ有効であるので，そのようなカットに対応する，特別の命令を用意している。

(2) 選択肢の発見とスタックの縮小： 通常の述語呼出しに先立つカット処理では，除去すべき選択肢の発見は容易である。即ち，非決定的であれば，最新の選択肢を除去すれば良い。これに対し，図4に示すような場合，述語"q" の実行により選択肢が生成されている（かもしれない）ので，発見の方法は単純ではない。また，最後の述語呼出しにおける，環境の除去の際に，除去された選択肢もスタックから取り除くには，若干の工夫が必要である。ＰＳＩ－Ⅱでは，以下の方法を用いている。

(a) 環境の中に，その環境が生成された時点における，最新の選択肢へのポインタと，クローズが決定的に選択されたか否かの情報（即ち"DET" ）を格納する。
(b) カット処理では環境に格納された"DET" を判定し
  (i) 決定的であれば，ポインタが指示する選択肢を最新の選択肢とする。
  (ii)非決定的であれば，ポインタが指示する選択肢の直前に生成された選択肢を最新のものとする。また，選択肢を除去したことを，環境の中に記憶する。
(c) 環境の除去の際に，カットにより除去された選択肢があれば，それも一緒に除去する。

3．クローズ選択

クローズ・インデクシングの対象となる引数が変数であるようなクローズや，同一の引数を持つクローズが存在する述語においては，高速なクローズ選択が必ずしも容易ではない。ＰＳＩ－Ⅱでは，いくつかの典型的な構造を持った述語に対して，クローズ選択を高速化するメカニズムを用意している。

(1) 'Until' Loop Optimaization： 図1に示した述語"nth_elem"の第２クローズは，第１引数が'0' でないという条件で，決定的に選択できる。一般に，クローズ・インデクシングの対象となる引数が変数であるようなクローズが，最後のクローズのみであるような場合（"Until" 型のループはこの例である），そのクローズ以外はすべて失敗することをあらかじめ判定することは，比較的容易である。"nth_elem"では，命令"jump_on_non_same_constant Ai,C,Lab"を用いて高速化することができる。この命令は'Ai'が'C' とユニファイ可能でない場合（即ち'C' でも変数でもない場合）'Lab' へ分岐する。"nth_elem"のループ一回当りの実行速度は，この最適化を行わない場合（図5(a) ）は４９step，行った場合（図5(b) ）は１６stepと見積もられ，約１：３の性能差がある。

```
  try_me_else  C2          jump_on_non_same_const
{1st clause codes}                A0,0,C2a
C2:                        try_me_else  C2
  trust_me_else fail       {1st clause codes}
{2nd clause codes}         C2:
                             trust_me_else fail
                           C2a:
                           {2nd clause codes}
        (a)                        (b)
```
図5

(2) Neck Cut Optimization ： "nth_elem"を図６のように変形すると，クローズ選択はさらに困難になる。また，図７のような例も同様である。さて，これらの述語は，次のような特徴を持っている。

(a) 引数の情報だけでは，決定的なクローズ選択ができない。
(b) 最後のクローズ以外は，通常の述語呼出しに先立つカット(Neck Cut)がある。

このような述語（またはクローズ群）に対応する選択肢に関しては，記憶すべき情報が少なくて済む。即ち，グローバル・スタックとトレイル・スタックのバックトラック点と，バックトラック時に実行すべきクローズのアドレスのみで良い（なお，カットの実行までは，引数レジスタの破壊や，永久変数の使用を行わないようにコードを生成することができる）。また，このような選択肢にのみ関わるトレイル情報を区別して，別のスタックに記憶すれば，カット処理におけるトレイル情報の選別除去を，極めて容易に行うことができる。図６の"nth_elem" のループ一回当りの実行速度は，この最適化を行わない場合は３８step，行った場合は２６stepと見積もられ，約１：１．５の性能差がある。

```
nth_elem(N,[_|L],E):-
      N =\= 0,!,
      N1 is N - 1,
      nth_elem(N1,L,E).
nth_elem(_,[E|_],E).
```
図６

```
p(X,...):-
  var(X),!,...
p(X,...):-
  atomic(X),!,...
p(X,...):-
  functor(X,F,A),...
```
図７

◇文献

[1] Warren,An Abstract Prolog Instruction Set, SRI Techinical Report 309
[2] 西川，ＰＳＩの性能評価（１），情報処理学会第３０回全国大会