ＫＬ１コンパイラにおけるコード生成の最適化

木村康則(1)　関田大吾(2)　近山隆(1)
(1)（財）新世代コンピュータ技術開発機構
(2)（株）三菱総合研究所

1．はじめに

現在，ＩＣＯＴでは，第５世代コンピュータプロジェクトの一環として並列推論マシンＰＩＭの研究開発を進めている。ＰＩＭ上の言語は，ＧＨＣに基づいた並列論理型言語ＫＬ１を予定している。本稿では，ＫＬ１ソースプログラムから抽象機械語［１］への効果的なコンパイル手法，特にクローズインデキシング手法について報告する。

2．クローズインデキシング

ＫＬ１は，ＧＨＣ(Guarded Horn Clauses) に基づく並列論理型言語であり，そのソースプログラムは以下の様に記述される。

```
a(X11,...X1j) :- g11, g12,...,g1k | b11,...,b1n
a(X21,...X2j) :- g21, g22,...,g2l | b21,...,b2o
                      :
a(Xi1,...Xij) :- gi1, gi2,...,gim | bi1,....bip
```

受動部　　能動部

図．１ ＫＬ１ソースプログラム

いま，あるゴール a(Y1,...Yj) が与えられると，このゴールは以下の様に実行される。

(1) 候補節群の中から１つ候補節を取り出し，ヘッドのユニフィケーション，ガード部の実行が試みられる。この時，ゴール変数を具体化しようとしたり，ガードの組込述語の実行に失敗すると，この候補節に対する選択の試みは失敗する。
(2) もし，(1) の実行が成功したならば，能動部のゴールが新たに実行可能ゴールとなり，これらのゴールの実行が同様に試みられる。
(3) 失敗ならば，別の候補節に対して (1) が行われる。
(4) 全ての候補節に対して失敗したならば，このゴールの実行は不可能であり，他ゴールによって（ゴールの変数が具体化することなどによって）実行可能になるまで実行が中断させられる。

ＫＬ１の言語仕様では，候補節の選択の順番は決められておらず，どの候補節から行ってもよいことになっている。従って，最も素直に候補節群をコンパイルするには，ソースプログラム上先に出てきたクローズから順にコードを生成し，最後にサスペンション時の処理を行う命令を置けばよい。しかし，この方法によると，ｉ番目のクローズの選択が失敗したことにより，ｉ＋ｊ番目（ｊ＞０）のクローズの選択も失敗することが分かってしまう場合でも実行が行われてしまう。これは，非常な実行時間の損失である。そこで，コンパイル時に分かる情報により，無駄な実行を抑え，実行が中断してしまうゴールを出来るだけ早く（実行時に）検出するようなコード列を生成することが望まれる。幸い，ＫＬ１では，Prolog と異なり，ソース上に書かれたクローズの順番と実行順序を保証する必要がないこと，ガード部のユニフィケーションが一方向性であることより，クローズのインデキシングは，比較的簡単に行えることが期待できる。

3．処理の概要

本節では，ＫＬ１コンパイラに実装したクローズインデキシングの方法について説明する。まず，基本的な方針を挙げる。
(1) インデキシングの対象とする引数は，各クローズのヘッドの第１引数からとし，各クローズが一意に決まるまで第２，第３引数に対象を拡げていく。
(2) 引数のデータ型は，整数，アトム，リスト，ベクタ，変数と，変数であるがガード部で或る決まったデータ型が来ることが期待されるもの（例えば，a(X, Y) :- Z := X + Y | p(Z). の様なクローズの第１，第２引数は整数でなければならない。）に分類する。
(3) インデキシング対象の引数がベクタの場合には，まず，その要素数で分類し，それで決まらない場合には，その要素をインデキシングの対象とする。（リストの場合もリスト要素をたぐってインデキシングの対象とする。）
(4) インデキシング対象の引数のデレファレンス結果は，引数レジスタに書き戻し，デレファレンス回数を１回に抑える。

この様な方針のもとで，コンパイラの処理手順の概略を示すと次の様になる。
(1) まず，同じ述語名及び引数を持つクローズを一纏めにして扱い，各々のクローズのヘッド引数に対して，そのクローズが選択されるための引数の条件を求める。図．２の例の場合は以下の様になる。

Efficient code generation scheme in KL1 compiler
Yasunori KIMURA, Daigo SEKITA(*) and Takashi CHIKAYAMA
Institute for New Generation Computer Technology(ICOT)
(*) Mitsubishi Research Institute(MRI)

```
クローズ(1)    < atom([]), integer(10) >
クローズ(2)    < [[atom(f),any]|any], any >
クローズ(3)    < [[atom(g),INT]|any], INT >
```

ここで，atom([]) は，アトム'[]' が実引数として与えられなくてはならないことを，同様に，'[a, b]'は2引数ベクタで，その要素が 'a','b' であること，'INT'は，任意の整数であることを示す。'any' は何でもよいことを示す。

(2) 次に，この条件を基に，各ノードが引数の条件による分岐，リーフ（葉）が分類されたクローズに対応するトリーを作る。この時，各ノードまでの分類で分かった引数の条件も各ノードに持たせておく。

(3) その後，このトリーをたぐって往きながらクローズのコンパイルを行う。各ノードがインデキシングの命令に対応し，クローズのコンパイルでは，そこまでのトリーたぐりによって既に分かった引数に対するコードは生成しないようにする。図．2のプログラムをコンパイルした結果を図．3に示す。

```
a([], 10) :- true | true.                    (1)
a([f(X)|Cdr], Y) :- true | p(Y).             (2)
a([g(X)|Cdr], Y) :- add(X, Y, Z) | p(Z).     (3)
```

図2．サンプルプログラム

```
a/2:    try_me_else  a/2/0
        switch_on_type  A1, a/2/1, a/2/2, a/2/0
a/2/1:  test_constant  [], A1    /* クローズ(1) */
        wait_constant  10, A2
        proceed
a/2/2:  read_variable  X3        /* リストの分解 */
        read_variable  X4
        jump_on_non_vector  X3
        test_arity  2, X3        /* ベクタの引数   */
        read_variable  X5        /* チェックと     */
        read_variable  X6        /* 要素の読み出し */
        jump_on_non_constant X5
        switch_on_constant X5, [(f, a/2/5),(g, a/2/6)]
a/2/5:  put_value  A2, A1        /* クローズ(2) */
        execute  p/1
a/2/6:  integer  X6              /* クローズ(3) */
        integer  A2
        add  X6, A2, A1
        execute  p/1
a/2/0:  suspend  a/2
```

図．3 コンパイル結果

4．インデキシング用の命令

本節では，ＫＬ１コンパイラの出力するインデキシング用命令の一部について説明する。その他の命令については，文献［１］を参照されたい。

(1) switch_on_type Ai, Atomic, List, Vector

Ai をデレファレンス後，そのデータタイプに依ってそれぞれの分岐先に分岐する。'Atomic' はAiが整数かアトムの時の分岐先である。Aiがリストかベクタの時は，分岐する前にＳＰ（ストラクチャポインタ）で第１要素を指させる。

(2) switch_on_arity Ai, [(N0, L0),...(Nk, Lk)]

Ai の引数個数（N0, N1,... Nk）に依って分岐先L0,... Lk に分岐する。Ai はデレファレンス済で，ベクタを指していなければならない。

(3) switch_on_constant Ai, [(C0, L0),...(Ck, Lk)]

Ai の内容（C0, C1,... Ck）に依って分岐先L0,...Lk に分岐する。Ai はデレファレンス済で，整数かアトムでなければならない。

(4) test_constant Cnst, Ai

Ai がCnst と等しいかどうか調べる。Ai はデレファレンス済で，整数かアトムでなければならない。

(5) test_arity Arity, Ai

Ai の引数個数がArity かどうか調べる。Ai はデレファレンス済でベクタを指していなければならない。

(6) jump_on_non_XXX Ai

Ai のデータタイプを調べ，それが 'XXX' で指定したものと同じならば成功，さもなくば失敗する。Aiはデレファレンスされる。この命令は，switch_on_type命令で分岐先が二方向しかない場合の最適化命令である。

5．まとめと今後の課題

現在，ＩＣＯＴで開発した逐次マシン上のＫＬ１処理系（ＰＤＳＳシステム［２］）に実装し，評価を行っている。ＢＵＰ（ボトムアップパーザ）やＫＬ１で書いたＫＬ１コンパイラを実行した時には，実行時間で，１～２割の向上が得られている。Prologに較べて，インデキシングの効果は小さい。これは，①ＫＬ１のガード部の実行では，呼出し側引数を具体化することがないので，トレイルの必要がないこと，②ＭＲＢ［３］によるＧＣのために，引数レジスタを壊さないようにコンパイルしているので，引数を退避しなくてよい，などの理由で，ガード実行の失敗時のコストが比較的軽いためである。

今後は，さらにインデキシング命令の最適化と，ガード部の組込述語の実行の重複を避けるようなコンパイル方法について検討を進める予定である。

謝辞

日頃御指導頂くＩＣＯＴ４研，内田俊一室長に感謝します。実装に関し，貴重なコメントを下さった宮崎敏彦(ICOT)，平野喜芳(SSL)，中越靖行(SSL) の各氏に感謝します。

参考文献

［１］Y.Kimura and T.Chikayama : "An Abstract KL1 Machine and its Instruction Set", Proc. of SLP'87

［２］宮崎，木村："並列論理型ＯＳ －ＰＩＭＯＳ（２）開発支援ＫＬ１処理系"，第35回情処全大(昭和62年後期)

［３］T.Chikayama and Y.Kimura : "Multiple Reference Management in Flat GHC", Proc. of ICLP'87