7Y-6

LRCのためのオブジェクト・コード

関田 大吾
((株) 三菱総合研究所)

木村 康則
((財) 新世代コンピュータ技術開発機構)

1 はじめに

LRC(Lazy Reference Count) 方式GCは、比較的低コストにループ以外の使用したメモリを完全に回収することが出来るインクリメンタルGC方式である。LRCをメインテナンスする機械語命令を生成するには、コンパイラによって静的に各プログラム節を解析し、変数参照数の増減を調べることが必要である。

我々は、並列論理型言語KL1でのLRC用命令セットを出力するコンパイラを開発した。本稿では、その命令セット、及び、その出力のための変数参照数解析方法について説明する。

2 LRCメインテナンス用命令セット

LRC用の命令セットは基本的にコミット直後に構造体要素をガードで読み出す時に参照数を増加する命令、ボディで参照数を増加する命令、参照数を減少する命令の3通りに分類できる。以下の各命令語引き数に'Ax'とあるのは変数を表し、'x'は、各変数に一意的に割り当てられた数である。

2.1 構造体要素読み出し時に参照数を増加する命令

構造体要素をガードで読み出すことにより、要素への直接の参照ポインタが新しく設けられたときに出される命令で、add_ref_elementが該当する。

```
p([X|Y]) :- true |...

wait_list              A2, A1
read_variable          A3
read_variable          A4
add_ref_element        A2, 0, A3
add_ref_element        A2, 1, A4
collect_list           A2
...
```

add_ref_elementの3つの引き数は各々、構造体ポインタ、要素番号、要素ポインタを表す。

2.2 ボディで出される参照数を増加する命令

ボディで出される参照数を増加する命令には、set_variable, put_variable, write_variable, set_value, put_value, write_value, add_referenceが含まれる。これらの命令語では、各変数に対する参照数の増加を命令の引き数として与える必要がある。

```
p(X) :- q(X, Y), r(X, Y).

wait_variable          A2, A1
create_goal            r/2
set_value              A2, G1, 1
set_variable           A3, G2, 2
enqueue_goal           r/2
put_value              A2, A4, 0
put_value              A3, A5, 0
execute                q(A4, A5)
```

上の例で、set_value, set_variable, put_value命令の第3引き数が各々の第1引き数で表される変数の参照数の増加を示す。

2.3 参照数を減少する命令

参照数を減少する命令には、collect_value, collect_listが含まれる。これらの命令は、ガードでのみ使用されている変数について出力される。

```
p(X):-true|true.

wait_variable          A2, A1
collect_value          A2
proceed
```

この例では、ヘッドに現れている変数'X'はボディで出現しない。これは変数'X'に対する参照数の減少を示しており、collect_list命令を出すことにより参照パスを1つ消費させる。

3 基本方式

本コンパイラは、図1のようなKL1B命令[3]を一度生成した後に(1)各変数の参照数の変化を調べ、(2)参照数変化に応じて命令の生成、変換を行っている。

以下で各々の手続きについて説明する.

3.1 参照数変化算出

参照数変化算出手続きでは、2.2に分類された命令語の出力のための準備を行う。

ボディでの参照数変化算出の際注意しなくてはならないのは、内部的に単にデータの受渡しにのみ使用されている変数は、参照数の変化には何等寄与しないという点である。

図1中の線はデータの流れを表している。この線が分岐する時にデータの参照数が増加したことになる(図1中●)。このデータの流れの分岐はソース側オペランドの変数がその以降の命令で参照されているかどうかに依って決定することができる。例えば、(1)の引き数に現れる変数A3は、(3)で再び使用されており、参照数がこ

Object Code for LRC
Daigo Sekita (Mutsubishi Reserch Institute, Inc.)
Yasunori Kimura(ICOT)

ソースコード

```
a([X|Y]) :- b(X, Z), c(X, Y, Z).
```

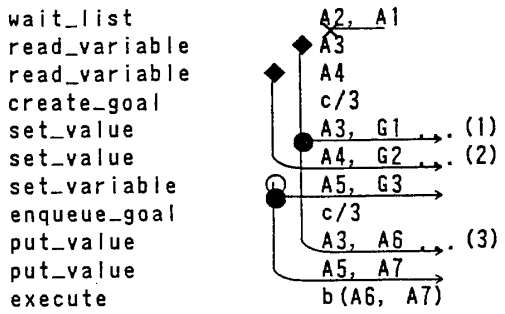

図 1: LRC メインテナンスをしない命令

こで 1 増加していると考えられる。一方、(2) の引き数に現れる変数 A4 は以降の命令では使用されておらず、ここでの参照数の増加はないことが判る。

また、図1中 A5 はソース・コード中の Z に対応しているが、このようなボディで初出の被複写変数は、参照数が 0 から 1 へ変化する (図1中○)。

以上で示したように、ボディでの変数参照数の増加の判断には、その変数がその前後でも現れているかどうかという情報が不可欠である。本コンパイラでは、前もって変数の出現範囲 (ライフ・タイム) を登録したテーブルを作成し、参照数変化の判定を行っている。

3.2 命令語変換・生成

第 2 節で示した命令語分類に従い、命令語変換・生成手続きを以下に説明する。

3.2.1 構造体要素読み出し時に参照数を増加する命令

構造体要素読み出し時に参照数を増加する命令生成のため、本コンパイラでは、前もって各変数が構造体要素であるか否かを登録しておき、ガードで変数を読み出す命令語 (read_XXX,wait_XXX) の引き数が構造体要素であれば対応して命令を出力している (図1中◆)。但し，要素が atomic なデータであることが判っている時はこれを抑制する。

3.2.2 ボディで出される参照数を増加する命令

先に算出された参照数の増加を基に各命令語の参照数フィールドを生成する。

- set_XXX 系,write_XXX 系,put_XXX 系

  第 1 引き数の参照数の増加をそのまま出力する。但し、put_value,put_variable については第 1 引き数、第 2 引き数参照数増加の和を出力する.

- get_XXX 系

  各引き数変数の参照数が増加していることが確認されれば、add_reference 命令をその変数に対して出力する.

```
wait_list              A2, A1
read_variable          A3
read_variable          A4
retrieve_list_element  A2, A3, A4
create goal            c/3
set_value              A3, G1, 1
set_value              A4, G2, 0
set_variable           A5, G3, 2
enqueue_goal           c/3
put_value              A3, A6, 0
put_value              A5, A7, 0
execute                b(A6, A7)
```

図 2: LRC メインテナンス用命令

3.2.3 参照数を減少させる命令

これは，ガードにのみ出現する変数に対して出力する (図1中×)。ただし atomic なデータについては出す必要はない。

また、出現頻度の高い命令パターン

```
add_ref_element  A1, 0, A2
add_ref_element  A1, 1, A3
collect_list          A1
```

に対して、このコード列を最適化した命令語

```
retrieve_list_element A1, A2, A3
```

が用意されている。これは、一度前者のコードを生成した後マージ可能な組合せを捜し出し、変換を行う。以上で述べたアルゴリズムを用いて生成されたコードは図2のようなものになる。

4 今後の課題

参照数のカウントのためには、データ・フローの解析が必要であることが解った。更に綿密な解析を行い、クローズ・インデキシング等の最適化技法も併用することによりいっそう良質、高速なコードが生成できると考える。

5 参考文献

(1)K.Ueda “Guarded Horn Clauses:A Parallel Logic Programming Language with the Concept of a Guard” ICOT Technical Report TR 208, ICOT, 1986.

(2)T.Chikayama,Y.Kimura “Multiple Reference Management in Flat GHC” 4th ICLP, 1987.

(3)Y.Kimura,T.Chikayama “An Abstract KL1 Machine and its Instruction set” SLP’87.

(4)A.Goto et al. “Lazy Reference Counting : An Incremental Garbage Collection Method for Parallel Inference Machine” Proc. of ICSLP’88.