KL1 クローズインデキシング方式の評価

6Q-6

西崎慎一郎[1], 平野喜芳[1], 武井則雄[1], 森田京子[1], 木村康則[2]
1: (株) 富士通ソーシアルサイエンスラボラトリ
2: (財) 新世代コンピュータ技術開発機構

1 はじめに

KL1 は、ICOT で設計された並列論理型言語である。KL1 のクローズインデキシング方式[1] は、クローズが選択されるためのガード部の条件をコンパイル時に調べ、効率化したコードを出力し、実行速度を向上させるものである。本稿では KL1 のクローズインデキシング方式の評価結果について報告する。

なお、本研究は第5世代コンピュータプロジェクトの一環として行なわれたものである。

2 クローズインデキシング方式の概要

コンパイル時に、同じ述語名、引数個数を持つクローズ群を一まとめにして扱う。これらのクローズの選択の順番は任意である。

そこで各クローズが選択されるための引数の条件およびガード部の組込述語を調べ、以下の方針でコンパイルする。

1. ゴールの引数の状態により、実行の中断、失敗が分かる場合は、できるだけ早くそれを見つける。
2. 複数のクローズのガード部で以下のような同じ処理がある場合、それらをまとめ、重複した処理をやらないようにする。
   - デレファレンス及び変数かどうかのチェック
   - データタイプのチェック
   - 値のチェック
   - 構造体の分解
   - 組込述語

2.1 インデキシング用の KL1-B 命令

インデキシング用にガード部の処理を単一化した機能を持つ KL1-B 命令[2] を以下のように設定した。

1. デレファレンス命令
   - wait Ai
     Ai をデレファレンスし、未定義なら try_me_else で設定された次候補へ分岐する。それ以外は次命令。
2. タイプチェック命令
   - is_atom Ai, LFail
     Ai(デレファレンス済み) のデータタイプがアトムでなければ、LFail へ分岐する。それ以外は次命令。同様な命令に is_integer, is_list, is_vector, is_string がある。
   - switch_on_type Ai, LA, LI, LL, LV, LS, LFail
     Ai(デレファレンス済み) のタイプ別に LA,... へ分岐する。どのタイプでもなければ LFail へ分岐する。
3. 値チェック命令
   - test_constant Const, Ai, LFail
     Ai(アトムまたは整数) が Const と等しくなければ、LFail へ分岐する。それ以外は次命令。同様な命令に test_arity がある。
   - branch_on_constant Ai, [(C1,L1),(C2,L2),...,LFail]
     Ai(アトムまたは整数) が Cn と等しければ、Ln へ分岐する。どれとも等しくなければ LFail へ分岐する。同様な命令に branch_on_arity がある。

実行の効率化には、これらの命令を組み合わせればよいが、実行命令数が多くなりやすい。そこで最適化のため、幾つかの命令をマージした命令を設定する。

4. デレファレンス＋タイプチェック命令 (1)
   - atom Ai
     サスペンド先と失敗先が同じ。分岐先は try_me_else で設定された次候補。同様な命令に integer, wait_list, deref_vector, wait_string がある。
5. デレファレンス＋タイプチェック命令 (2)
   - jump_on_non_atom Ai, LFail
     サスペンド先と失敗先が違う。タイプチェックの失敗時は LFail へ分岐。同様な命令に jump_on_non_integer, jump_on_non_list, jump_on_non_vector, jump_on_non_string がある。
6. タイプチェック＋値チェック命令
   - check_constant Const, Ai, LFail
     同様な命令に check_vector がある。
7. デレファレンス＋タイプチェック＋値チェック命令
   - wait_constant Const, Ai
     分岐先は try_me_else で設定された次候補。同様な命令に wait_vector がある。

3 評価

以下に実際にクローズインデキシングをかけた場合 (以下 ON) とかけない場合 (以下 OFF) についての評価結果を示す。

評価に使用したプログラムは、以下の4種類である。

prime :: 素数生成プログラム
qlay :: 8-Queen 解法プログラム
bup :: ボトムアップパーザ
kl1cmp :: KL1 自身による KL1 コンパイラ

Performance evaluation of KL1 clause indexing.
Shin'ichirou NISHIZAKI[1], Kiyoshi HIRANO[1], Norio TAKEI[1], Kyouko MORITA[1], Yasunori KIMURA[2]
1: Fujitsu Social Science Laboratory Ltd. 2: Institute for New Generation Computer Technology (ICOT)

3.1 オブジェクトコードの静的データ

表1に静的なオブジェクトコードのサイズ(行数)の違いを示す。ONでは処理が最適化されたため、サイズが小さくなっている。

	OFF	ON	±(%)
prime	137	114	−16.8
qlay	250	201	−19.6
bup	3400	3233	−4.9
kl1cmp	8666	8060	−7.0

表1: オブジェクトコードのサイズ

3.2 実行時の動的データ

実行は、KL1エミュレータPDSS[3]上で行なった。実行時間はミリ秒単位で、5回の平均である。ただし、タイマの精度等から考えて100ミリ秒未満の数値の違いはあまり意味がない。

表2、表3にOFFとONでの変化を示す。

qlayは4割程度の速度向上があり、最低でも1割程度の速度の向上があった。インデキシングの効果が現れたといえる。

	OFF	ON	±(%)
prime	1940	1720	−11.3
qlay	12830	8010	−37.6
bup	12470	10740	−13.9
kl1cmp	6430	5830	−9.3

表2: 実行時間

	OFF	ON	±(%)
prime	105499	95268	−9.7
qlay	722799	433202	−40.1
bup	581555	464622	−20.1
kl1cmp	292642	255501	−12.7

表3: 実行命令数

次に、ガード部で分岐する可能性がある命令の実行回数(Instr.)と、実際に分岐した回数(Jump)は、表4のようになる。

ONでは、分岐命令実行数、分岐回数が減っており、ガード部の効率化の目的は果たしているといえる。

	OFF		ON	
	Instr.	Jump	Instr.	Jump
prime	41989	596	27792	501
qlay	280832	54563	121439	24967
bup	121640	75294	73850	24302
kl1cmp	92674	12377	65835	10624

表4: ガード部の分岐命令実行数、分岐回数

4 インデキシングの効果の分析

実行結果を基に、インデキシングの効果の特徴が現れている幾つかの例を示す。

1. クローズの数が多い場合

   各クローズに共通な処理が行なわれている場合が多いため、インデキシングの効果が大きい。

   - 定数を待つクローズが続く場合

     OFFでは、wait_constant命令で定数をクローズが書かれた順に逐次にサーチしていく。しかし、ONでは定数を一まとめにしてbranch_on_constant命令で分岐するため、速度が向上する。

     【例】 bupでは、以下のようなテーブルを幾つか扱っている。アトムを扱う命令の実行数が全体で84271 ⇒ 42813と減少した。

```
dict(R,P,ga)        :- true | P=p,R=t.
dict(R,P,icot)      :- true | P=pro_noun,R=t.
dict(R,A,atarashii) :- true | A=adj,R=t.
        :
```

   - ガード部が複雑な場合

     失敗時に同じ処理を繰り返さないように最適化されている。

     【例】 qlayでは、ガード部でネストしたリストを扱っている。リストを扱う命令の実行数が全体で74548 ⇒ 31116と減少した。

```
filter([[I|_]|Ins], I,K,Out) :- true     | ...
filter([[J|_]|Ins], I,K,Out) :- K := I-J | ...
filter([[J|_]|Ins], I,K,Out) :- K := J-I | ...
filter([[J|In]|Ins],I,K,Out) :- I =\= J, ...
```

2. 単独のクローズで、ガード部の条件がない場合

   無駄なtry_me_elseを出さないように最適化されている。

3. 互いに背反な組込述語をがある場合

   どちらか一方だけ実行するように最適化されている。

   【例】 primeではequalとnot_equalが背反になっている。ONではnot_equalのみ実行しているので、equalの実行数が404 ⇒ 0となった。

```
filter(P, [X|Xs1], Ys0) :- X mod P =\= 0 | ...
filter(P, [X|Xs1], Ys0) :- X mod P =:= 0 | ...
```

4. 選択されるクローズが偏っている場合

   OFFでは、書かれた順にクローズが候補になる。実行時に先の方に書かれたクローズが多く選択される場合は、ON/OFFはあまり影響がない。しかし後の方が多く選択される場合は、OFFでは無駄な処理が多くなる。一方、ONでは、各クローズを平均化することになるので、結果として効率がよくなる。

   クローズの選択される確率を意識していないプログラムや、実行時に動的に変化する場合などに、効果が期待できる。

   【例】 primeのgenという述語は、上のクローズがほとんど選択され、下は一度しか選択されない。このクローズの順番を入れ替えると、ONでは変化しない。しかし、OFFでは命令実行数が105499 ⇒ 108487と増え、実行速度も遅くなる。

```
gen(N0, Max, Ns0) :- N0 =< Max | ...
gen(N0, Max, Ns0) :- N0 >  Max | ...
```

5 今後の方針

命令ごとの実行回数、分岐回数等の検討を進め、インデキシング命令を更に最適化する予定である。また、今回の評価に使用したプログラムは、ほとんどサスペンドをしないので、サスペンドの多いプログラムを使った評価も行なう予定である。

謝辞

日頃御指導頂いているICOT第4研究室の方々に感謝します。

参考文献

[1] 木村、関田、近山: KL1におけるコード生成の最適化, 情報処理学会第36回全国大会 7H-4,1988

[2] 木村 他: KL1抽象命令セットの改良について, 本大会予稿集

[3] PDSS — 言語仕様と使用手引 — ,ICOT TM-437,1988