ＧＨＣプログラムの最適化

森田正雄*、吉光宏*、太細孝**、上田和紀***

（株）三菱総合研究所　**三菱電機（株）　***ＩＣＯＴ

１．はじめに

我々は汎用計算機上にＷＡＭ［Warren83］をベースとしたＧＨＣ［Ueda86］の処理系を開発している［森田86］。本処理系はガード部を解析して、節の選択を決定木に展開する最適化手法を取り入れている［森田87］。

本処理系でＧＨＣプログラムをコンパイルすると、手続き型言語とはかなり違った機械コードが出力される。比較・条件分岐命令が非常に多い機械コード列になっている。

これは、手続き（同じ頭部を持つ節の集合）という小さな単位（手続き型言語のサブルーチンや関数という単位よりはるかに小さな単位）で独立にコンパイルしているため、実際の処理の部分と比べて、その手続きに渡ってくるデータの全ての可能性を考慮した動的な検査（比較・条件分岐命令）を行う部分の比重が高くなってしまっているからである。そしてこれは、プログラム実行上のかなりのオーバヘッドになっている。

このような非効率的な処理を大域的な視点で最適化する方法を述べる。これにより、外部データに対する動的な検査の比重の高かった小さなループ中の処理が効率よく処理された。

２．動的な検査部分の分析

比較・条件分岐命令を多く出している主因は、単一化コードである。単一化時、次のような検査を動的に行っている。

(1)　中断・再開始検査
(2)　ポインタのたぐり操作
(3)　モード検査（ボディでの構造体単一化時）
(4)　ヒープ領域あふれの検査

(1)の中断・再開始検査は、ガード部での単一化時ゴール側を具体化するか否かを検査する中断検査とボディでの単一化時変数の具体化を待っているゴールがあるか否かを検査する再開始検査とがある。

(2)のポインタのたぐり操作は、単一化時レジスタ値がポインタかを動的に判定し、ポインタであれば、それをたぐり寄せ、値に行き着くまで繰り返す操作である。

(3)はボディでの構造体単一化時、双方向の単一化かまたは一方向の単一化かを動的に判定する検査である。

(4)は単一化時動的にデータが生成される領域（ヒープ領域）のあふれ検査であるが、静的なプログラム解析による最適化は難しく、本稿では触れない。

３．最適化の方法

実行時の動的な検査を最適化するため、次のような方法をとる。

コンパイルの単位をある程度大きくし、トップ・レベルのゴールだけ外から見えるようにし、それ以外の手続きを外部から参照できない局所的な手続きとする。このようにすることにより、コンパイル単位のトップ・レベルから渡ってくる情報を直接参照する単一化は、あらゆるケースを想定した検査が必要であるが、それ以外の単一化は静的な解析により、多くの動的な検査を排除することができる。

例えば、図１のようなプログラムを例にとると、

```
(1)  n([]    ,O) :- true | O = [].
(2)  n([H|T],O) :- true | n(T,O1), a(O1,[H],O).

(3)  a([]    ,I,J) :- true | I = J.
(4)  a([I|J],K,L) :- true | L = [I|M], a(J,K,M).
```

図１　リストの反転

ｎ／２をトップ・レベルの手続きとし、ａ／３を局所的な手続きとすると、ｎ／２の第一引数はどのようなものか特定できないが、ａ／３の第一引数はｎ／２内部で生成されるものが渡ってくるため、動的な検査の一部を必要としない。

さらに外部の影響を取り去るため、手続き間の呼び出し関係から、入力（ガード部で検査されるもの）と出力（ボディの単一化で値がバインドされ、出てくるもの）に引数の分類（モード推論）を試み、トップ・レベルのゴールの出力引数を別の未定義変数に置き換え、その未定義変数と出力引数との単一化をその下位のゴールがすべて終了または中断した後実行されるようにプロセス・キューにつなぐ。また、中断から再開始する場合にも、中断したゴールの出力引数を別の変数に置き換え、その未定義変数と出力引数との単一化をプロセス・キューにつなぐ。

このようにすることにより、出力引数に対する単一化は、動的な検査の不要な代入となり、ゴール呼び出し時出力引数が未定義変数であった場合に効率よく処理される。

なお現状、この最適化はゴールの引数が入力と出力とに分類できることを前提に最適化しているため、プログラムによりこの最適化が行えないものがある。

３．１　中断・再開始処理と検査

単一化での動的な検査の最適化とともに、コンパイル単位内での不要な中断・再開始処理をなくすことも重要な最適化の一つである。

中断・再開始処理の回数を減らす方法として、１つのデータ（ストリーム）を複数のプロセスが待つ場合に、待ちと駆動のきっかけを１回にする方法と、データ生成する側とデータを消費する側との間で、常にデータを生成する側を先行させ、停滞（中断・再開始処理）を起こさないよう

Optimization of GHC programs.

M.MORITA*, H.YOSHIMITSU*, T.DASAI**, K.UEDA***
*MRI, **MELCO, ***ICOT

にする方法とがある。ここではプログラム変換を要しない後者についてのみ行う。

ＧＨＣの並列性を逐次計算機で模擬する場合、処理系側がボディ・ゴール中のゴール呼び出し順決定の自由を持っている。コンパイル時この自由度を利用して、中断・再開始処理が少なくなるような呼び出し順を決定する。（ボディ・ゴール中の共有変数の入出力関係を調べ、出力引数としているものを先に呼ぶ。例えば図１の例で、（２）のボディ・ゴールはｎ／２を先に実行し、ａ／３を後で実行する。）

このように中断・再開始処理の最適化を行った後、中断・再開始処理が起こり得るところにのみ、中断・再開始検査を行い、それ以外の単一化の処理から、中断・再開始検査部分を取り去る。

３．２　ポインタのたぐり操作

計算機上で論理変数を表現する場合、ポインタの助けを借りる。そのため単一化時、毎回ポインタかを判定し、ポインタであれば、たぐり寄せ、値に行き着くまでこれを繰り返す。この操作は、一見論理型言語と手続き型言語の実行時の不可避な壁のように見える。しかし、データの出生が明らかであれば、多くの動的なたぐり操作を取り除くことができる。

例えば図２のプログラムで、

```
(1)  a :- true | b(X), c(X).
(2)  b(X) :- true | X = foo.
(3)  c(X) :- X = foo | true.
```

図２

（１）を実行すると、変数Ｘの変数セルが生成される。そして（２）を実行するとき、引数レジスタは変数セルへのポインタとなる。未定義変数の場合、変数セルへのポインタが値となるため、ｂ／１のボディでのたぐり操作が不要である。そして（２）の実行により変数セルに値がセットされる。次に（３）を実行するとき引数レジスタは（１）て呼び出されたときの値の入った変数セルへのポインタである。従ってｃ／１のガード部での単一化時１回のたぐり操作が必要である。

このように単独で変数セルができる場合、出力側のボディの単一化ではたぐり操作は不要で、入力側のガードの単一化では１回のたぐり操作が必要とする。また基底項が渡る場合は入力する側のたぐり操作は不要である。

図１のプログラムを例にとると、ｎ／２の第一引数は外部からのストリームを参照しているため、最適化できない。しかし、ａ／３はコンパイル単位内で生成したデータしか参照していないため、最適化が可能である。ａ／３の入口て第一引数の１回のたぐり操作が必要で、それ以降ＴＲＯループ内では、ガードでもボディでもたぐり操作はいらない（ｎ／２の出力引数は実行時点で分離されているため、意識しなくてよい）。

３．３　モード判定

ボディての構造体単一化時のモード判定を最適化する。これは、３．で述べたような解析で、出力引数と判定されたものをwrite モードとし、write モードの処理のみオブジェクトに展開する。それ以外は動的なモード判定を行う。

４．簡単なプログラムでの評価

リスト反転と自然数の生成で評価を行ったところ、表１のような結果となった。

表１　ＶＡＸ１１／７８０での評価

プログラム	最適化	性能	命令実行回数
n-reverse 300 要素	無し	33.2	27
	有り	52.9	18
generator 1～40000	無し	21.1	40
	有り	42.5	22

（注　性能の単位はＫｒｐｓ、また命令実行回数は頻度高く実行されるＴＲＯループの命令実行回数（ n_reverseであればappendのＴＲＯループ内の命令実行回数）

append,generatorのガードでのたぐり操作、ボディでのたぐり操作、モード判定などが最適化されている。

表１で命令実行回数が減る割合以上に性能が向上している理由は、動的な条件分岐命令が減ることにより、ハードウェアからみて、不規則な条件分岐命令が減り、命令先読み実行等の最適化が効力を発揮したためと推察される。

５．おわりに

ＧＨＣのプログラムの最適化は、動的な細かい検査を排し、仕様レベルの並列性を満たす範囲内で、いかに効率のよい逐次処理に置き換えるかというところに行き着くと考えられる。このような視点に立ち、最適化の具体的な方法を提示した。

また並列計算機上でもこの最適化技法は、他のプロセスを意識せずに動作できる処理部分で流用できる。

今後、実用に向けて幅を広めて行きたいと考えている。また、現状の処理系の処理方式の面でも、一般のＧＨＣプログラムが逐次処理に置き換えられる度合が十分大きければ、スタック・ベースの処理方式に移行することも考えたい。

なお本研究は、第五世代プロジェクトの一貫としてＩＣＯＴの委託で行ったものである。

《参考文献》

[Warren 83] Warren, D.H.D.:
" AN ABSTRACT PROLOG INSTRUCTION SET "
SRI International, Technical Note 309

[Ueda 86]　Kazunori, Ueda:
" Guarded Horn Clauses "
LNCS 221, Springer

[森田 86] 森田、吉光、太細、上田:
汎用計算機上のＧＨＣ処理系
第３３回情処全国大会　6D-2

[森田 87] 森田、吉光、太細、上田:
汎用計算機上のＧＨＣコンパイラ
第３５回情処全国大会　5Q-4