4Y-1 バクチ方式によるコンパイラ

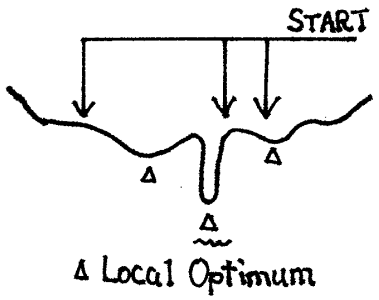
梅村恭司（ＮＴＴソフトウェア研究所）

おおまかなバックトラック（バクチ）を利用し、コンパイルの速度と翻訳コードの速度を両立するコンパイラが作成できた。コンパイルの速度を低下させるように考えられているバックトラック技法であるが、おおまかなバックトラックは良好な性質をもたらす。

１）背景

良質な翻訳コードを生成するには、Control Flow GraphとData Dependency Graphの解析しプログラムの構造を変形して処理する方法[1]、生成されたコードを処理して不用な転送命令や分岐命令を省くピープホール最適化の方法[2]などが知られている。これらは最適化をしないコンパイラにオプショナルパスを追加することで実現している。パスを追加する方法では、レジスタ割付方針や呼び出しの方針などの根本的な部分に手を加えて最適化を実施することは困難である。このような方法では、あるコード生成の方針に沿った範囲での最適コードはもとまるが、生成方針が根本的に異なる範囲にある真の最適コードはもとまらない。

一方、エキスパートシステムなどの実現から、ローカルな最適解と真の最適解の問題が検討されている。決定的な方法は知られていないが、真の最適解に近づけるための工夫がいくつか存在する。そのなかで、異なったルールセットをパラレルに適用して解を求める方法（パラレルソルバー）がある。ここで行った方法はコンパイラの処理にパラレルソルバーと等価な方法を使ったものである。



図：局所解への落ち込み

２）概要

コンパイラは独立した３つのコードジェネレータを持っている。ひとつは関数の実行にレジスタの待避が不要であるという仮定を持って生成するもの。ふたつめは、ループのなかでレジスタを破壊するようなプリミティブ以外の機能を使わないという仮定を持って生成するもの。最後は、仮定を置かないもので、すべての場合に対し安全にコードを生成するものである。はじめの二つのジェネレータは不完全なもので、仮定に適合しないものは処理をすることができない。しかし、うまく処理できることに賭けて翻訳処理を始める。そして、処理ができないことを発見した時点で、次のジェネレータがコード生成をやり直す（バックトラックする）ことになる。

ジェネレータは逐次に起動されるが、この問題にはパラレルに起動したのと等価である。まず、コード生成の問題では、一つの方法で解の有無は速やかにわかる。つまり、ひとつのジェネレータは有限時間内に空集合か解を含む集合のどちらかを求めることができる。一方、この問題では最初に得られる解を全体の解と考えてよい。最初には強い仮定を置いているので、最初には都合のよいアルゴリズムが使える。それゆ

A gambling Compiler
Kyoji UMEMURA
Nippon Telegraph and Telephone Corp.

え、ひとたび解が求まればそれ以後のジェネレータで更によい解がもとまる確率は小さい。コード生成は逐次的に実行されるが、得られるものはパラレルに実行したものと等価である。

３つのコードジェネレータは、コードの生成方針がまったく異なる。ひとつめは扱うデータ、変数や式の演算結果のほとんどをレジスタに割り付けようとする。ふたつめはループ内部で使われるデータをレジスタ割り付けようとする。最後のものは中間結果以外はレジスタに割り付けない。３つのプログラムは共有部分のないものとなっている。

３）仮定の有効性

仮定に適合するような例が多いことを示す。仮定が成立するようなプログラムが、かなりの確率で存在するので、仮定をもつ特別なコードジェネレータを作る意味がある。一番強い仮定である、レジスタ待避の不要という仮定を満たす例を示す。

```
(defun memq (x y)
    (cond ((atom y) y) ((eq x (car y)) y) (t (memq x (cdr y))) ))
```

ここで、atom, eq, car, cdr, condなどはレジスタ上だけで実行できる操作である。memqはレジスタを破壊する可能性があるが、実行最後の呼び出しであるので引数を整えて分岐すればよい。このように、引数の型判定をして分岐する形式のの関数はレジスタを待避しなくてもよいものの一つである。このような関数は、nreverse、assqやその変形に適用できる。

４）バックトラックと翻訳速度、

大まかなバックトラックの方法では翻訳時間はプログラムの大きさに対して直線的に比例する。フロー解析とは異なり、変数が多くても遅くならない。いままでコンパイラの処理にバックトラックなどの非決定性を取り込むことが行われなかった理由は、翻訳速度が問題となるからである。しかし、３つのジェネレータに大まかなバックトラックを適用したときは最悪でも３倍の時間がかかるだけである。現実には前処理、後処理があるので、時間の増加率はさらに少なくなる。

入力プログラム長との相対時間ばかりでなく、翻訳の絶対時間も極端に長くなることはない。市販のコンパイラと比較して良好な結果が得られた。例えば上記memqの定義を今回のコンパイラで翻訳するのに0.047secである。同じ機械上でのVAXLISPでは0.060secであった。そして、生成されたコードで長さ10000のリストを検索するのには、今回のコンパイラの翻訳結果では0.043secであり、VAXLISPでは0.160secであった。この結果は、ほとんど同じ翻訳時間でよい翻訳コードが生成できたことを意味する。

まとめ

グローバルな最適値を求めるパラレルソルバーを翻訳処理に適用し、大まかなバックトラック（バクチ）で実現した。この場合、翻訳速度を大きく低下させることなく速度の速い翻訳コードが得られることを実証的に確かめた。

参考文献

[1]Jeanne F., Karl J. O. and Joe D. W.:"The Program Dependence Graph and Its Use in Optimization," ACM TPLS, Vol 9, No. 3, July 1987.

[2]Aho A. V. and Ullman J. D.:"Compilers: Principles Techniques and Tools.", Addison Wesley, 1977.