マシンコード変換による最適化コンパイラ

4Y-2

藤波順久（東京大学）

1 はじめに

移植可能コンパイラは通常、マシンに依存しない中間コードを生成し、それからコード生成部でマシンコードを生成する。移植のときにはコード生成部だけを書き換える。ところが、最適化されたコードを生成するためには、コード生成部で多くの場合分けが必要で、しかもマシンにより異なるので、移植は実際には容易ではない。マシンに依存しない最適化を中間コードに対して行うこともあるが、中間コードが複雑になって、コード生成部の書き換えはさらに困難になる。

コード生成後に、マシンの記述に従って覗穴的最適化（peephole optimization）を行うものもある[1,2,3,5]。オプティマイザ自体はマシン非依存だが、特定のマシン向けのコンパイラに匹敵する性能がでる[2]。このようにした場合の利点は、コード生成部は単純な中間コードから単純なマシンコードへの変換でよいこと、マシンに依存した共通部分式など、コード生成前には発見できないような最適化を行えることである。

しかし、マシンコードに対する最適化では、以下で述べるように、通常あまり表に出て来なかった問題を含んでおり、既存のオプティマイザでは、適当な仮定をおいて問題を避けている。ここでは、現在製作中の、マシンコードの最適化を行うコンパイラの概要を述べ、問題点と解決方法を示す。

2 コンパイラの構成

コンパイラのソース言語は手続き型言語（複数）を想定している。コンパイラの前半の部分では、ソース言語にもマシンにも依存しない、スタックマシン的な中間コードを出力する。出力はコード生成部でマシンコードに変換される。マシンコードに対しては、基本ブロックの最適化、覗穴的最適化、大域的最適化などを行うが、何回か繰り返す必要があるので、各最適化の入出力のデータ表現は、すべてマシンコードを用いる。

[2,3]ではスタックのシミュレートのためにレジスタが無限にあると仮定して、最適化後に実レジスタに割り付けているが、レジスタによって最適化が異なるマシンに対応するため、レジスタ割り付けも最適化中に行うようにした。

マシンコードの表現は、[1,2,3]のように、レジスタ、メモリ、フラグなどの総称であるセルへの代入を用いる。例えば、R0に0番地の内容を転送する命令はR[0]←M[0]と表せる。ジャンプ命令はプログラムカウンタPCへの代入で表す。

3 マシンコード最適化の問題点

通常の最適化は、変数の定義と使用を分析して行う。マシンコードでは、変数とはセルのことになるが、ローカル変数や配列変数など、間接参照をするものは、同じセルに対する参照かどうかの判定が困難である。[3]では、ふつうは変数の参照方法は1通りで、他の参照により変更された可能性のあるときは、必ずコンパイラの前半で値を無効にする擬似命令を入れると仮定している。しかし、このような仮定は危険である。

そこで、間接アドレスを示すセル（例えばフレームポインタ）の値に条件をつけることによって、重なりのある可能性のあるセルを限定することにした。例えば、ローカル変数への代入がグローバル変数を変更しないのは、フレームポインタがグローバル変数の番地から十分離れているからである。このことは、スタックチェックを行っていれば、そのコードから推定できるし、行っていなくても、条件を擬似命令として入れておけばよい。条件が不明のときは、任意の値をとり得ると仮定するので、最適化によってプログラムの意味を変えることはない。

配列変数の場合にも、添え字の範囲チェックのコードや範囲を示す擬似命令によって、重なりを限定できる。帰納変数（inductive variable）の解析によって添え字の範囲が推定できることも多い。最適化によって添え字が明らかに範囲からはみ出すことがわかった場合、コンパイル時エラーとすることもできる。

コンパイラが作りだした一時変数をポインタなどで参照するのは、言語仕様上できないはずである。つまり、参照方法は1通りである。このような場合、参照方法が1通りであることを示す擬似命令を用いる。

このように、通常は最も危険な場合を仮定しておけば、多少は最適化をしそこなう場合があるが、安全である。

Optimization through Machine Code Transformation

Nobuhisa FUJINAMI

The University of Tokyo

4　文脈について

擬似命令を導入すると言ったが、擬似命令を最適化にどのように反映させるか、正確に定義しておかなければならない。そのために、文脈という概念を使う。

マシン命令の置き換えが正しいかどうかは、文脈に依存することが多い。[4]では、左文脈と右文脈を用いている。左文脈とは、そのマシン語命令を実行する時点において値が等しいことがわかっているセルの組の集合、右文脈とは、そのマシン語命令より後では値が使用されないことがわかっているセル（死んでいるセル）の集合である。ここでは、左文脈を拡張して、セルの値に関する論理式の集合とする。つまり、左文脈が｛“$c_1=v_1$”，…，“$c_n=v_n$”｝であるとき、$c_1=v_1$&…&$c_n=v_n$という論理式が成り立っていると考えるのである。命令の置き換えの前後で、右文脈に含まれないすべてのセルに同じ値が入ることが左文脈から証明できるとき、その置き換えは正しい。

ある命令c←vについての左文脈をLとするとき、次の命令についての左文脈は、（L－｛cと同じ可能性のあるセルについての条件｝）∪P（｛“c＝v”｝）とする。P（K）は、LとKから導出できる条件を表す。これらはすべてである必要はないが、その分最適化が行われなくなる。[3]では、最も古いセルの値を使って標準形を作るといううまい方法を用いている。セルがcと同じ可能性があるかどうかは、Lから判定する。ただし、参照方法が1通りであることを示す擬似命令が有効な間は、同じ参照方法のとのみ、同じ可能性があるとする。

条件ジャンプ命令PC←if f then LABEL else PCのときには、次の命令は2つあるが、条件fが成り立った場合、成り立たなかった場合それぞれについて、左文脈はL∪P（｛“f”｝）、L∪P（｛“not f”｝）とする。

セルの値の条件を示す擬似命令があったときには、それを左文脈に加える。

ある命令c←vの右文脈をRとするとき、その前の命令の右文脈は、もしc∈RならR、そうでなければ（R∪｛c｝）－｛cやvで参照されている可能性のあるセル｝とする。

例を示そう。“if i<10 then a[i]:=0 j:=i endif”に対するi8086のマシン語は、

```
        MOV AX,[BP+4]                   AX←M16[BP+4]
        CMP AX,10                       C←AX<u16 10 etc.
        JAE L0                          PC←(if C=1 then L0 else PC)
*1      MOV SI,AX                       SI←AX
        SHL SI,1                        SI←SI<<1 etc.
*2      MOV WORD PTR [BP+SI-20],0       M16[BP+SI-20]←0
*3      MOV AX,[BP+4]                   AX←M16[BP+4]
        MOV [BP+6],AX                   M16[BP+6]←AX
L0:
```

（ただし、AX,BP,SIはR[0],R[5],R[6]の略、M_{16}はワードメモリ、$<_{u16}$は符号なしワード比較）のようになるが、*1の時点での左文脈は、“C=1”の古い形である“($M_{16}[BP+4]<_{u16}10$)=1”を含んでいるため、$M_{16}[BP+4]<_{u16}10$であることを証明できる。すると、*2ではBP-20以上BP未満の番地のセルしか変更できないことがわかる。そのため、“AX=M_{16}[BP+4]”という左文脈は消去されないので、*3の命令を除去することができる。

5　おわりに

筆者らは、すでにMS-DOS上のBCPLコンパイラのオプティマイザを製作し、性能を評価している。このオプティマイザは、局所的なパターン変換と、ループの最適化を行っており、実行速度が2倍以上になったプログラムもある。この際、ループの最適化が特に重要であることがわかっている。

今後は、以上の結果を利用してコンパイラ、特にコード生成部以後を試作し、性能を評価することによって、マシン依存性が低く、言語仕様で許されているもの以外はプログラムの意味を変えないで、しかも性能のよいオプティマイザを製作する方法を示す予定である。

【参考文献】

(1) Davidson, J.W. and Fraser, C.W.: The Design and Application of a Retargetable Peephole Optimizer, ACM Trans. Program. Lang. Syst., 2,2 (April 1980), pp. 191-202.

(2) Davidson, J.W. and Fraser, C.W.: Code Selection through Object Code Optimization, ACM Trans. Program. Lang. Syst., 6,4 (October 1985), pp. 7-32.

(3) Davidson, J.W. and Fraser, C.W.: Register Allocation and Exhaustive Peephole Optimization, Software - Practice and Experience 14,9 (September 1984), pp. 857-866.

(4) Giegerich, R.: A Formal Framework for the Derivation of Machine-Specific Optimizers, ACM Trans. Program. Lang. Syst., 5,3 (July 1983), pp. 478-498.

(5) Kessler, R.R.: Peep - An architectural Description Driven Peephole Optimizer, SIGPLAN Notices, 19,6 (June 1984), pp. 106-110.