素朴なレジスタプロモーションの実装と評価

Implementation and Evaluation of a Simple Register Promortion

狩野　祐介 † 佐々　政孝 †
Yuusuke Kanou Masataka Sassa

† 東京工業大学　大学院情報理工学研究科　数理・計算科学専攻
Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
{kano2, sassa}@is.titech.ac.jp

プログラム中の値をメモリからレジスタに移しても意味が変わらないような部分では、メモリの値をレジスタに移してメモリアクセスの回数を減らすことができる。この、メモリ変数を一時的にレジスタに格上げする最適化をレジスタプロモーションという。本研究では、特にグローバル変数を対象とした比較的素朴なレジスタプロモーションを COINS （並列化コンパイラ向け共通インフラストラクチャ）上で LIR （低水準中間表現）の最適化部として実装、評価した。

1 はじめに

現代の RISC 型計算機のコンパイラは通常レジスタ割り当てという処理を行って、プログラム中のスカラー (集合体でないもの) 変数を可能な限りレジスタに乗せる。しかし、一部のスカラー変数はレジスタ割り当ての候補に入っていないため、仮にレジスタが無限に使えたとしてもメモリアクセスが少なからず残る。

C 言語において素朴なレジスタ割り当てを仮定すると、次の二つはレジスタ割り当ての候補から外れる。

1．メモリアドレスを使用される変数

2．グローバル変数

両者ともアクセス元が明確でないために、レジスタの値を維持できない場合があり、精密な解析をしない限りレジスタに乗せることはできない。例えば、一つの値のために別名が存在したら、それは定義するごとにメモリーに保存しなくてはならないし、それぞれ使う前にメモリーからロードしなくてはならない。しかし、これらの変数もアクセス元や別名に関する情報が明確な範囲ではレジスタに格上げすることができる。この最適化がレジスタプロモーションである。

本研究では、Keith D. Cooper と John Lu の論文「Register Promotion in C Programs」[1] で紹介されたアルゴリズムをもとに、レジスタプロモーションを実装した。

本論文の構成は以下の通りである。

第 2 節では「Register Promotion in C Programs」で紹介されたアルゴリズムを例を使って説明し、実装に対応した変更点を述べる。第 3 節では、実装する環境の説明を行う。第 4 節では、実装したものを使ってレジスタプロモーションを行った結果を比較、評価する。第 5 節では、まとめと今後の課題を述べる。

2 アルゴリズム

2.1 レジスタプロモーションとは

プログラム中の値をメモリからレジスタに移しても意味が変わらないような部分では、メモリに入っていた値をレジスタに移したほうが実行効率が上がる。このようなメモリ変数を一時的にレジスタに格上げするような最適化を **レジスタプロモーション**(レジスタ促進) という。レジスタプロモーションは主にループを対象としており次に例を示す。

```
loop{                    r=a;
  r=a;                   loop{
  r=r+1;      ==>          r=r+1;
  a=r;                   }
}                        a=r;
  (A)                      (B)
```

これはループ内で `a=a+1` という計算を行う例で、(A)(B) はそれぞれ目的コードをソース風に示したものである。また、`a` はメモリを、`r` はレジスタを表している。目的コードをソース風に表示した (A) において、ループ内の `a` の値はループの反復一回ごとにメモリからロードされ、計算が終わったらメモリに

戻される。これを (B) のように、ループの直前で a の値をメモリからレジスタ r にロードし、ループ内の a の演算をレジスタ r の演算に置き換え、ループを出るとき r の値を a のメモリ位置に戻すように変更する。こうするとメモリアクセスがループの出入り口のみになり効率が上がる。

```
p=&a;
loop{                    loop{
  a=a+1;                   a=a+1;
  *p=1;                    f()
}                        }
  (C)                      (D)
```

(C)(D) は促進が行えない場合のソースプログラムの例を示したものである。(C) においては、a を促進しようとして前の例のように a をレジスタに置き換えると正しい最適化にはならない。それは*p が a を間接参照しているからである。(D) については、手続き f() を呼び出している。呼び出し先で a が使われる場合、呼び出し先の a はメモリを参照するので、このループの a をレジスタに置き換えると意味が変わってしまう。(C) のように間接的にメモリの値を書き換えるものが含まれたり、(D) のように手続きの呼び出し先でループ内の値が参照される場合は促進が行えないことがある。これらを定式化するため、メモリ位置への直接参照を「明確な参照」といい、メモリ位置への間接的な参照を「あいまいな参照」という。(C) の例では a=a+1 の a は明確な参照で、*p は a へのあいまいな参照である。(D) では、f() の中で a が現れるのなら f() も a へのあいまいな参照である。

2.2 アルゴリズム

Cooper らのアルゴリズムではレジスタプロモーションは以下のように進めていく。

まず手続き内においてループ構造（自然ループ）を見つける。ループ同士の包含関係も調べておく。

次にそれぞれの自然ループについて二つの集合を計算する。$\texttt{L\_Explicit}_l$ は自然ループ l 内の明確に参照されるすべてのタグの集合、$\texttt{L\_Ambiguous}_l$ は l においてあいまいな参照を受けるすべてのタグの集合である。なお、タグとはメモリ位置の字面上の識別名である。

そして、$\texttt{L\_Promotable}_l$、$\texttt{L\_Lift}_l$ の二つを求める。この二つを計算するデータフロー方程式を表 1 に示した。

$\texttt{L\_Promotable}_l$ はループ l で促進できるタグの集合であり、$\texttt{L\_Explicit}_l$ と $\texttt{L\_Ambiguous}_l$ から求まる。$\texttt{L\_Lift}_l$ はループ l で促進すべきタグの集合である。ループが二重以上になっている場合、同じタグであればできるだけ大きいループで促進するのが望ましい。小さいループに関して促進したときよりも、より多くの値をレジスタへの参照に置き換えることができるからである。なお *surrounding-loop(l)* とはループ l を囲むループのことである。

各ループについて $\texttt{L\_Lift}_l$ に含まれるタグそれぞれについて、仮想レジスタ v をつくる。さらにループ内におけるそれらのタグへの参照をすべて v への参照に変換する。仮想レジスタとはコンパイラが作る変数で、目的コードではレジスタに置き換えられるものである。

仮想レジスタを使うように書き換えられたタグに関しては、促進可能な最外ループに入る直前に新しいブロックを挿入し、そこで仮想レジスタにロードする。また、そのループから出るときは、出た直後に新しいブロックを挿入し、そこでもとのメモリにストアする。
これでレジスタプロモーションが終了する。

以上のことを例を用いて示す。図 1 の 2 つのフローグラフは左が促進前、右が促進後のものである。各ループについての情報は表 2 のようになり、L1 では a が、L2 では b が促進の対象となる。よって、L1 の入口と出口にそれぞれ新しいブロック NB1、NB4 を挿入する。そして NB1 にはロード命令 r1=a; を、NB4 にはストア命令 a=r1; をセットする。L2 に関しても同様に新しいブロックと命令をセットする。これでこのフローグラフのレジスタプロモーションが完成する。

Loop Information		
Loop	L1	L2
Blocks	B2-B5	B3-B4
L_Explicit	a,b,c	a,b
L_Ambiguous	b,c	c
L_Promotable	a	a,b
L_Lift	a	b

表 2: Loop Information

$$\texttt{L\_Promotable}_l = \texttt{L\_Explicit}_l - \texttt{L\_Ambiguous}_l$$

$$\texttt{L\_Lift}_l = \begin{cases} \texttt{L\_Promotable}_l & \text{if l is an outermost loop} \\ \texttt{L\_Promotable}_l - \texttt{L\_Promotable}_{surrounding-loop(l)} & otherwise \end{cases}$$

表 1: データフロー方程式

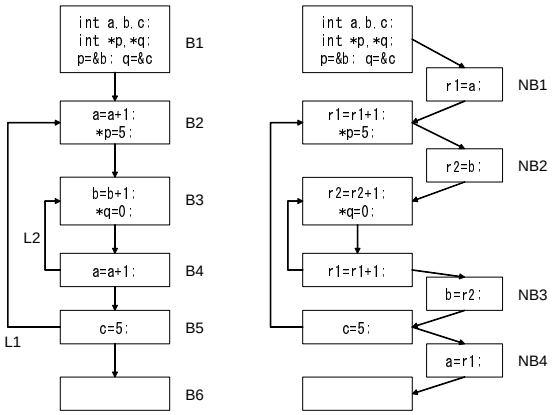

図 1: レジスタプロモーションの例

2.3 実装に対応した変更

冒頭でも述べたように実装は COINS 上で行ったが、それに起因して促進に制限を加える必要がある。COINS にはメモリにしまわれている値をレジスタに乗せる最適化を行う部分が既に組み込まれている。しかし、そこで対象としている値はローカル変数のみであり、あいまいな参照などの解析が必要ない範囲で適用されている。具体的には、おもにアドレスを取られていないローカル変数をレジスタに乗せている。(詳しくは COINS の仕様書のバックエンド部 [4] を参照）そこで本研究では、レジスタープロモーションの対象をスカラーのグローバル変数のみに絞った。

また COINS のバックエンドでは、手続きごとに最適化などの処理が行われるため、現在処理中の手続き以外の部分で設定された別名情報は得ることが困難である。よってある手続きに関してレジスタプロモーションを行うとき、その手続きの外部で得られる別名情報は一切ないことを前提とした。

これらの制限のもとでは、ループの中にあいまいな参照があればそのループでは一切促進が行えないということになる。あいまいな参照が起こりうるのは次の場合である。

(1) ループ l 内に手続き呼び出しがある場合
(2) ループ l 内にメモリへの間接参照がある場合

(1) については、手続き呼び出し元のループ l 内のグローバル変数の値が呼び出し先で参照される可能性があるので、ループ l 内で使われるタグはすべてあいまいな参照を受ける可能性がある。よって間違った促進を避けるために l 内のすべてのタグをあいまいな参照を受けるものとして扱わざるを得ない。

(2) については、実際は（いくつかの）特定のタグだけがあいまいな参照を受けている。しかし, 十分な別名情報が得られないため、ループ l 内のどのタグについてもこのあいまいな参照の参照先となる危険性が否定できない。例えば以下のようなソースプログラムの場合、

```
a=0;
p=&b;
f(){
  loop{
    a=a+1;
    *p=2;
  }
}
```

p と a はグローバル変数である。*p の参照先は b なので b だけがあいまいな参照を受けているのだが、手続き外の情報がないので f() の外にある p=&b という情報はないことになる。よって手続き内部だけでは*p の参照先が特定できないのである。このような場合も (実は他の場合もあるのだが)a を含むループ l 内のすべてのタグをあいまいな参照を受けるものとして扱わざるを得ない。

以上より (1)、(2) のいずれかに当てはまるループに関しては、ループ内のすべてのタグがあいまいな参照を受けるとして扱い、促進は行わない。

3 実装について

実装にはCOINSのバージョン1.1.2を採用し、LIR（低水準中間表現）の最適化部の最適化の一つとしてレジスタプロモーションを組み込む形で行った。前節でも述べたとおり、促進の対象はスカラーのグローバル変数のみとした。さらに対象言語は特にC言語とする。

4 実験結果と考察

本節では、実装したレジスタプロモーションの実験結果とそれに対する考察を行う。

実験は、Sun Microsystemsの Sun Blade 1000で行った。この主な仕様は、表3の通り。

Architecture	Superscalar SPARC Version 9
CPU	750MHz UltraSPARCIII × 2
Cache	64KB データ 32KB インストラクション
Memory	1GByte
OS	SunOS 5.8

表 3: Sun Blade 1000 の主な仕様

テストプログラムにはCOINSの開発やテストに使われているプログラムよりヒープソート (heap.c)、１４女王問題 (queen.c)、シェルソート (shell.c)、挿入ソート (InsertionSort.c) 選択ソート (Selection-Sort.c)、素数算出プログラム (tpprime.c) の６つと、SPEC CPU2000のベンチマークプログラムより、164.gzip, 197.parser, 254.gap, 256.bzip2, 300.twolf, 177.mesa, 179.art, 183.equake, 188.ammp の９つを用いた。レジスタプロモーション (促進) を行う場合と行わない場合で実験を行った。

COINSのテストプログラムについては実験の各項目においてそれぞれ５回ずつ実行し実行時間の平均を取った。SPECのプログラムについてはそれぞれ３回ずつ実行し、実行時間の中央値を取った。なおCOINSのテストプログラムの実行時間計測には、timeコマンドを用いた。なお、timeコマンドでは1,2%の誤差は意味を持たない。

COINSのテストプログラムについての実行時間は図2のようになった。COINSo0が、コンパイル時にオプション指定しなくてもかかる簡単な最適化のみのもの。COINSo0+RPが、それにレジスタプロモーションをかけたものである。

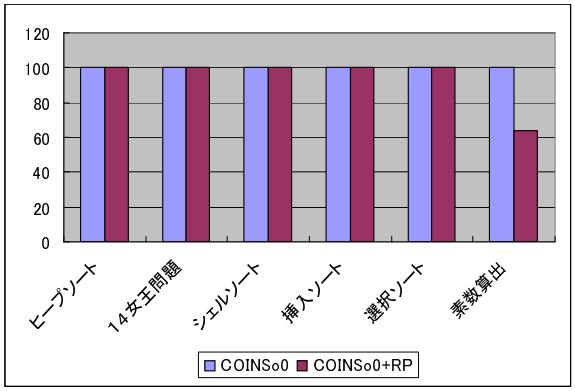

図 2: COINS のテストプログラムの実行時間 (COINSo0 に対する相対比)

ほとんどのプログラムでは目立った結果が得られなかった。これは、プログラム中で促進された値の数が少なく、また促進が行われたループも実行頻度の小さいものであり促進の効果が目立たなかったためと考えられる。素数算出においては、促進された値の数もループの反復回数も他のものより格段に多く、実行時間は30%以上短縮された。さらに素数算出をC言語コンパイラのgccでオプション-O2をつけてコンパイルして実行してみた。このオプションは、gccであらかじめ用意されているさまざまな最適化をかけるためのものでかなり強力な最適化がかかるが、実行時間はCOINSo0のおよそ72%となり、COINSo0+RPの方が10%ほど早いという結果になった（COINSo0+RPはCOINSo0の64%であった）。これは、レジスタプロモーションが、十分に適用されるようなプログラムではいかに強力な最適化であるかということを示しているといえる。

SPECのプログラムについての結果は図3のようになった。COINSo0がコンパイル時にオプション指定しなくてもかかる簡単な最適化のみをかけたもの。COINSo0+RPが、それにレジスタプロモーションをかけたものである。

いくつかのプログラムで1~5%程度の実行時間の改善が見られた。ログを調べたところ、改善がほとんどゼロに等しいものに関しては、促進された値の個数も著しく少なかった。300.twolfにおいては10%程度実行時間が長くなった。このプログラムにおいては大変多くの値が促進されているのだが、吐き出さ

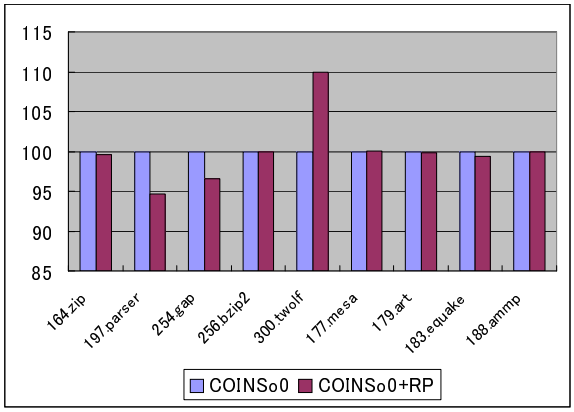

図 3: SPEC のベンチマークプログラムの実行時間 (COINSo0 に対する相対比)

れたコードを調べたところ、１つのループに対して最大で 20 個以上もの値を促進していた。このためレジスタ圧力が上がり過ぎ、スピルが起こったためにこのように実行時間が遅くなってしまったと考えられる。

5 まとめと今後の課題

総じて見ると、レジスタプロモーションは、実行頻度の高い部分に適用された場合には絶大な効果があることがわかる。その反面、一度に促進する値の個数が多すぎるとレジスタ圧力が上がりすぎ、実行時間を長くしてしまうという一面もある。本研究では、対象をグローバル変数のみに絞るなどいくつか制限を加えたため、レジスタプロモーションの適用範囲が狭まり、改善の見られないケースが多かった。

今後の課題として以下の３つが挙げられる。１つ目は、ポインタ解析をすることによって、ループ内に間接参照があっても促進が行える場合は行うようにすること。２つ目は、呼び出し先の手続き内まで別名解析することで、手続き間にまで拡張したレジスタプロモーションを行えるようにすること。３つ目は、一度に促進する値の数を制限することで、レジスタ圧力が上がり過ぎないようにすること。これらにより適用範囲を広げることが当面の課題となるであろう。

謝辞

本研究の一部は，文部科学省科学技術振興調整費，科学研究費，および財団法人栢森情報科学振興財団の補助を受けた．また実装にあたって、細かい質問にも丁寧に答えてくださった LSI Japan の森公一郎氏に感謝する。

参考文献

[1] Keith D. Cooper, John Lu: Register Promotion in C Programs. *PLDI '97: Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation*, ACM Press.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman *Compilers : Principles, Techniques, and Tools.* Addison Wesley, 1985.

[3] COINS Project. "COINS Project homepage.". http://www.coins-project.org/.

[4] 森 公一郎 "COINS Project Backend Part" http://soukou.cs.uec.ac.jp/~kmori/.