プログラミング言語処理系論 (10)
Design and Implementation of Programming Language Processors

佐藤周行
（情報基盤センター/電気系専攻融合情報学コース）

SSAを用いた最適化

- われわれは、SSAを作るときに以下のものを同時につくることができる。
  - DEF/USE chain and USE/DEF chainを、DEFひとつに対するUSEの集合として管理できる。

- Dead Code Elimination
  - v= x op y において、vのUSEが0であれば、この文は削除できる。
- Constant Propagation
  - v = x において、xのDEFが定数 x = cであればこれをv = cに置き換えることができる。
  - v = φ(c1,c2,…,cn)で、c1,…cnがすべて同じであればv=c1に置き換えることができる。

- Copy Propagation
- Constant Folding
- Constant Conditions によるjump文の最適化
- Unreachable Codeの除去による unreachable blockの除去

今回(と次回)の予定

- その他、重要な最適化について
- Interprocedural analysis/optimizations
- プログラム解析の進んだ方法論について (Effect)

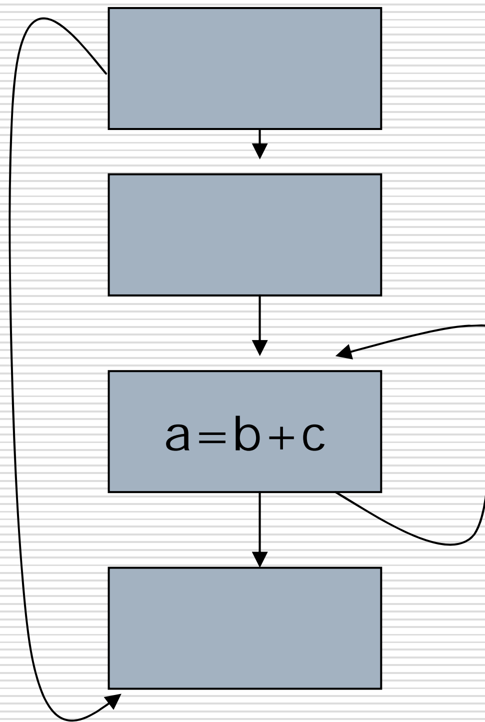
Loop Invariant

- Def Loop Invariant
  当該loop中での式の計算が常に同じ値を生むとき、その計算をloop invariantという。
- 例:
  L1
  cmp x,y
  jmplt L2
  a=1
  x=x+1
  jmp L1

- ループ中に式 t＝a1 op a2があるとせよ。次の条件が満たされるとき、この計算はloop invariantである。
(1) aiが定数である。
(2) aiのDefがループの外側にある。

Loop invariant hoisting

- Loop invariantはそのループの外に出せる



方法論はPREに含まれる。

PREの一環でなくても、Defがループの外にあるかどうかはSSA変換されているなら一瞬でわかる

左のようにしても、peephole最適化をして、ループの形を変えることができる。

Induction variables

- Def induction variables x
  In every loop iteration, x's value is incremented/decremented in constants.
- 例:

```
L1:
    i=i+1
    ...
    goto L1
```

Induction variablesの例

- ループのインデックス変数 (basic induction variables)
- Basic induction variablesから計算されるもの (derived induction variables)
  - 配列のアドレス計算
a[i] → a + i * 4
  - こちらは結構重要

Induction variable detection

- SSAを仮定する。ループ中から変数のDefをたどっていく。
- i3＝φ(i1,i2)
  …
  i2＝i3＋1

- たどっていって、φのうち、バックエッジから来るDefが、自分自身に定数を足したもの、もしくはそれと等価なものであればそれはbasic induction variableである。

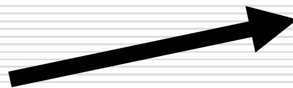
- Derived induction variable
  - Basic induction variableに線形計算を施して得られたものはderived induction variableである。

Induction variableの最適化

- 掛け算は足し算にできる(strength reduction)

```
for (i=0; i<N;i++) {
   a[i] = b[i+1];
}
```

```
for (i=0; i<N;i++) {
   p = a+4*i;
   q = b+4*i+1;
   *p = *q;
}
```



```
for (i=0, p=a, q=b+1; i<N; i++) {
   *p = *q;
   p = p+4;
   q = q+4;
}
```

Interprocedural Analysis

- 今までの最適化　ー関数コールが出てこない（手続き内解析と最適化）
- 現状の最適化　ーwhole program optimizationが普通に行なわれる
- 手続きをまたがった解析と最適化が求められる

手続き間最適化の必要な例

- Alias解析
  - 手続きを呼ぶと、そこでは何をしているかわからないので、保守的に考えれば、手続きが終了した時点ですべての変数は変更されていると考える。
  - 普通はそんなことはないが、それを確認するためには、手続き間の解析が必要。

- 仮引数と変数のaliasを調べる
- 仮引数が、手続き内でどれとaliasにあるかを調べる
- 呼び出し方によっては仮引数どうしがaliasにある可能性もある
  - 現在のプログラミング言語では、言語の定義の中で適当な仮定を置ける場合がある。

- C99
  - restrict *ptr;
  - 仮引数に使えて、他の仮引数とaliasにないことを宣言する
  - no-alias コンパイラオプション(コンパイラに上と同じ事を指定)
- Fortran
  - 誕生のときから、引数はすべてrestrictであるとされる

Fortranのaliasの扱い

- Fortranのaliasの解釈は
  - ポインタと言う概念が未成熟な場面では自然なもの
  - 一部のアーキテクチャ (vector architecture, SIMD系) では、aliasを仮定しなくてもよいということは、アーキテクチャを活かすコード生成に非常に適したものだった
  - Cでは、その解釈ができなかった。1980—2000年ころにかけてFortranコンパイラが大発展した理由は、別に過去の遺産を継承したことからきたのではない
  - そのかわり、Fortranプログラムのaliasの管理はプログラマに投げ出され、ときどき間違った結果に悩まされることになった

インライン展開

- 関数呼び出しを、呼ばれる関数の本体で置き換える
- 期待される効果
  - 呼ばれる関数のサイズが小さいときには、関数コールのオーバヘッドが削減されることが期待される（現在のインライン展開のベースにあるのはこの発想）
  - 関数の本体が、取り込まれることにより、さらなる解析と最適化が期待される（部分評価その他）

インライン展開の例

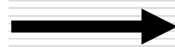
```
int f(int x, int y)
{
    return x+y;
}

int g()
{
    return f(1,2);
}
```

```
int f(int x, int y)
{
    return x+y;
}

int g()
{
    return 1+2;
}
```

Cloning

- 関数コールの引数の一部が定数の場合、それを展開して、新たな関数を作ることが有効な場合がある(部分評価の言葉では「特殊化」)。
- 例:

```
x = f(y, 2);

int f(int z, int w)
{
   if (w > 1) return g(z+2);
   else return h(z);
}
```



```
x = f2(y);

int f(int z, int w)
{
   if (w > 1) return g(z+2);
   else return h(z);
}

int f2(int z)
{  return g(z+2);
}
```

インライン展開の判断基準

- インライン展開＋αで性能が出るケースが非常に多い
  - 展開後の定数伝播
  - 展開後の条件式評価
  - 展開後のループ内での命令スケジューリング
- 従来の論文では、関数コールのオーバヘッド削減、コードサイズの増加率などが判断基準になっていた
  - 攻め方が間違っている

- 部分評価の可能性（cloningを含む）
- 展開後の他の最適化の適用可能性
- 上の基準に、引数の形がどのように関係しているか

従来のプログラミング言語での扱い

- マクロ展開（必ず展開）
  - C
- inline 修飾子（コンパイラはこれをヒントにしてよいという扱い）
  - C++, C

Effect

- プログラムの実行にともない、どのような効果が現れてくるのかを解析したい
  - Memoryのread/write/alloc
  - Binding time
  - Function call
  - Try/catch
  - …

実際の例

- 型システムに、注釈をつける形で表現される
- Def. Language
  e ::= c | x | fn_{π} x => e|e e|…
  τ ::= int | bool | … | $\tau \rightarrow \tau$

- 型に、以下のように注釈を加える
- $\psi ::= \{\pi\} \mid \psi \cup \psi \mid \phi$

  $\underline{\tau} ::= \mathrm{int} \mid \mathrm{bool} \mid \ldots\ \underline{\tau} \rightarrow_{\psi} \underline{\tau}$
- 注釈つき型を構成するときにどの関数が使われたかを憶えておく

- メモリのread/write/allocの解析

- ψ : effect
- ρ : region
- τ : annotated types

$$\psi ::= \{!\pi\} \mid \{\pi:=\} \mid \{\text{new } \pi\} \mid \psi \cup \psi \mid \phi$$

$$\rho ::= \{\pi\} \mid \rho \cup \rho \mid \phi$$

$$\underline{\tau} ::= \text{int} \mid \text{bool} \mid \ldots \mid \underline{\tau} \rightarrow_{\psi} \underline{\tau} \mid \underline{\tau} \text{ ref } \rho$$

- read/write/allocを次のように表現する

- !x: τ where type of x is τ ref
- x:=e: τ where type of x is τ ref
- new_{π} x:=e1 in e2 where …

- 以下のルールで e: τ & ψ を導出していく

$$\widehat{\Gamma} \vdash_{\mathsf{SE}} c : \widehat{\tau}_c \ \&\ \emptyset \qquad\qquad \widehat{\Gamma} \vdash_{\mathsf{SE}} x : \widehat{\Gamma}(x) \ \&\ \emptyset$$

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_x] \vdash_{\mathsf{SE}} e_0 : \widehat{\tau}_0 \ \&\ \varphi_0}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \mathtt{fn}_\pi\ x\ \mathtt{=>}\ e_0 : \widehat{\tau}_x \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \&\ \emptyset}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_2 \xrightarrow{\varphi_0} \widehat{\tau}_0 \ \&\ \varphi_1 \quad \widehat{\Gamma} \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1\ e_2 : \widehat{\tau}_0 \ \&\ \varphi_1 \cup \varphi_2 \cup \varphi_0}$$

$$\widehat{\Gamma} \vdash_{\mathsf{SE}} \mathtt{!}x : \widehat{\tau} \ \&\ \{\mathtt{!}\pi_1, \cdots, \mathtt{!}\pi_n\} \text{ if } \widehat{\Gamma}(x) = \widehat{\tau}\ \mathtt{ref}\ \{\pi_1, \cdots, \pi_n\}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau} \ \&\ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{SE}} x\ \mathtt{:=}\ e : \widehat{\tau} \ \&\ \varphi \cup \{\pi_1\mathtt{:=}, \cdots, \pi_n\mathtt{:=}\}} \text{ if } \widehat{\Gamma}(x) = \widehat{\tau}\ \mathtt{ref}\ \{\pi_1, \cdots, \pi_n\}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e_1 : \widehat{\tau}_1 \ \&\ \varphi_1 \quad \widehat{\Gamma}[x \mapsto \widehat{\tau}_1\ \mathtt{ref}(\varrho \cup \{\pi\})] \vdash_{\mathsf{SE}} e_2 : \widehat{\tau}_2 \ \&\ \varphi_2}{\widehat{\Gamma} \vdash_{\mathsf{SE}} \mathtt{new}_\pi\ x\ \mathtt{:=}\ e_1\ \mathtt{in}\ e_2 : \widehat{\tau}_2 \ \&\ (\varphi_1 \cup \varphi_2 \cup \{\mathtt{new}\,\pi\})}$$

$$\frac{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau} \ \&\ \varphi}{\widehat{\Gamma} \vdash_{\mathsf{SE}} e : \widehat{\tau} \ \&\ \varphi'} \text{ if } \varphi \subseteq \varphi'$$