Humboldt-Universität zu Berlin

Institut für Informatik

Codeoptimierung

SS 00

Übungsblatt 7
djloop – Gruppenarbeit (max. 4 Personen)

Abgabe: 18.7.00 vor 9:30h

(10 Punkte) *irloop* ermittelt die natürlichen und die unnatürlichen Schleifen eines Kontrollflußgraphen. Dazu sollen zunächst die unmittelbaren Dominatoren (idom) und damit der Dominatorbaum bestimmt werden. Die Dominatoren (doms) liegen bereits gemäß letzter Aufgabe vor. Darauf aufbauend sollen der DJ-Graph erstellt werden und sowohl die natürlichen Schleifen als auch die unnatürlichen Schleifen berechnet werden. Eine Liste der Blöcke einer natürlichen Schleife wird dabei mit dem Kopf der Schleife assoziiert, d.h. die Komponente "loops" der Blockstruktur ist genau dann nicht leer, wenn es sich um einen Kopf handelt. Analog werden unnatürliche Schleifen in *allen* Schleifeneingängen in "irloop" abgelegt. Die Blockinformationen werden bereits von dem Programm ausgegeben (zum Testen und Bewerten der Aufgabe). Man beachte, daß die erste Ausgabe erfolgt bevor Schleifen gefunden wurden (Kontrolle des DJ-Graphen, keine RTLs), während die zweite Ausgabe nach Finden der Schleifen erfolgt (Kontrolle der Schleifen, nicht jedoch des DJ-Graphen, der inzwischen durch Reduktion modifiziert worden sein kann, diesmal mit RTLs). Weiterhin wird der Flußgraph bereits für jede Funktion erstellt. Zur Bearbeitung der Aufgabe loggen Sie sich bitte auf einer Sun SPARC ein und geben (in Ihrem Home-Verzeichnis) die folgenden Befehle ein:

```
mkdir ~/codeopt
mkdir ~/codeopt/ub7
cd ~/codeopt/ub7
cp ~mueller/classes/codeopt/ub7/* .
```

Erweitern Sie dann die Datei *djloop.c* (das C Programm). Verändern Sie dazu nur die bisher nicht ausgefüllten Fuktionen. Testen Sie das Programm wie üblich, stellen Sie z.B. den Kontrollfluß mittels der Nachfolger/Vorgängerinfos graphisch dar und vergleichen Sie Ihre Ergebnisse mit dem Graphen.

```
ub1.cc ~mueller/test/src/cal.c   # erzeugt Assembler-Code "cal.s"
djloop < cal.s > cal.S           # erzeugt optimierten Assembler-Code "cal.S"
ub1.ld cal.S                     # erzeugt ausf"uhrbares Programm "cal.out"
cal.out 1995                     # Programm ausf"uhren
```

Die Abgabe umfaßt die Datei *djloop.c* und erfolgt per Upload durch http://eiche.informatik.hu-berlin.de:8081/codeopt bis zum oben genannten Datum.

Unnatürliche Schleifen

- Lösung: DJ-Graphen (Dominator/Jump-Graph) :=
  - Knoten der Flußgraphen,
  - Kanten des Dominatorbaumes (D-Kanten) plus
  - Rückwärtskanten (BJ-Kanten back-jump)
  - kreuzende Kanten (CJ-Kanten cross-jump): Kanten des Kontrollflußgraphen, die nicht D-/BJ-Kanten sind
- Tiefensuche über Graphen bildet deckenden Baum, in dem Kanten $x \to y$ wie folgt bezeichnet werden können:
  - sp-back Kante, falls $y = x$ oder y ist Vorgänger von x im deckenden Baum
  - sp-tree Kante, falls x ist Vater von y im deckenden Baum
  - sp-forward Kante, falls x ist Vorgänger aber nicht der Vater von y im deckenden Baum
  - sp-cross Kante, falls x und y keine Vorgängerbeziehung haben
- Algorithmus zum Finden aller Schleifen (nat. und unnat.):

```
 0: Stufe (Level) aller Knoten im Dominator bestimmen (0..NumLevel-1)
 1: Tiefensuche "uber DJ-Graphen, sp-back Kanten markieren
 2: for (i = NumLevel - 1 downto 0) /* Knoten von unten nach oben besuchen */
 3:   IrreducibleLoop = False;
 4:   foreach Knoten n mit n.level = i do
 5:     foreach Vorg"anger m von n (Kante m --> n) do
 6:       if Kante m --> n ist sowohl CJ-Kante als auch sp-back then
 7:         IrreducibleLoop = True; /* n ist in einer unnat"urlichen Schleife */
 8:       endif
 9:       if Kante m --> n ist BJ-Kante then
10:         Finde alle Schleifenknoten (siehe nat"urliche Schleife)
11:         Reduziere alle Schleifenknoten zu 1 Knoten (Kopf) im DJ-Graphen
12:       endif
13:     endfor
14:   endfor
15:   if (IrreducibleLoop) then /* es gibt >= 1 unnat"urliche Schleife */
16:     Finde SCCs auf Stufe >= i, jeder SCC (> 1 Knoten) ist eine Schleife
17:     Reduziere Knoten der Schleifen (von Zeile 14) zu 1 Knoten im DJ-Graphen
18:   endif
19: endfor
```

- Algorithmus zum Finden der SCCs eines Graphen:

```
 1: Tiefensuche "uber Graphen, Knoten bekommen Posttraversierungnummer finish
 2: Tiefensuche ab Knoten mit gr"o"ster finish-Nummer absteigend im
      inversen Graphen (Kanten umgekehrt -- Vorg"anger statt Nachfolger),
      wobei jeder gefundene Unterbaum ein SCC ist
```

- man beachte: SCC Algorithmus wird nur für die Kanten mit Stufe $\geq$ level im Algorithmus zum Finden aller Schleifen angewendet

Eliminierung gemeinsamer Teilausdrücke

- Algorithmus für lokale CSE:

  for each RTLs in the Block do

  1. Zerteile RTL: $dst = src$;
  2. Finde älteste s-expr mit Match für src, ersetzte Registervorkommen in src mit dem ältesten Wert in der Register-Äquivalenzklasse, Ergebnis ist $csrc$.
  3. Finde Adreßberechnung in dst (genannt $addr$). Finde älteste s-expr mit Match für $addr$, ersetzte wie zuvor, Ergebnis ist $caddr$. Benutze $caddr$ in dst, Ergbnis ist $cdst$.

     ```
     M[r[1]<<2] = ...;  addr == r[1]<<2
     ```

  4. Finde einfachste Ersetzung für src (bzw. $addr$) in Äquivalenzklasse von $csrc$ (bzw. $caddr$). Prüfe, ob neuer RTL legal und neue Kosten geringer. Wenn ja, führe Ersetzung für RTL durch und passe Liste toter Register an (altes Register in Liste wird ggf. durch neues ersetzt).
  5. Anpassung der Äquivalenzklassen: Lösche alle Vorkommen $cdst$ und Klasse $cdst$. Füge Klasse $cdst$ zur Klasse von $csrc$ hinzu.

- iterativer Algorithmus für globale CSE (ähnlich Drachenbuch S. 765, verbesserter Algorithmus S. 772)

```
Eingabe: e_gen[] und e_kill[] f"ur jeden Block
Ausgabe: in[] und out[] f"ur jeden Block

in[B1] := 0;
for each Block X do
  out[X] := 0;
while change of any out[] do
  for each Block X do
    in[X] := out[first_P in Preds(X)];
    for each other P in Preds(X) do
      in[X] := in[X] intersect out[P]; /* Schnittmenge */
    out[X] := e_gen[X] + (in[X] - e_kill[X]);
```