CSCI 565 Compiler Design

Fall 2015

Homework 4

Due Date: November 20th, 2015 (electronically)

Problem 1: Control-Flow, Loops and Loop Optimizations [40 points]

Consider the code depicted below for a function with integer local variables i, a, b, c and parameters of the enclosing procedure p1, and p2.

```
00: i = 0
01: a = 0
02: b = 1
03: c = 1
04: L1: if (p1 = 0) goto L3
05: L2: i = i + 1
06: a = 0
07: goto L4
08: L3: a = 1
09: if (p2 = 0) goto L6
10: return
11: L5: i = i + 1
12: if (i < 32) goto L2
13: goto L3
14: L6: A[i] = a
15: goto L5
16: L4: a = 0
17: c = b * 4
18: A[c] = a
19: if (i < 32) goto L2
20: b = 1
21: c = c + 1
22: i = 0
23: goto L6
```

For this code fragment determine:

a. [05 points] The Control Flow Graph (CFG).
b. [05 points] The dominator tree and the loop(s) (identify the back edges).
c. [10 points] Use algebraic properties and copy propagation to improve the execution of the body of the loop(s) identified in (b).
d. [10 points] Based on the code obtained in (c) recognize loop induction variables and transform the code to eliminate them as much as possible. Indicate which variables are basic induction variables and which are derived induction variables.
e. [10 points] Discuss if there are opportunities for loop invariant code motion in the original code versus the transformed code obtained in (d). Argue about the correctness of your transformations.
Problem 2: Scalar Replacement and Loop Peeling [25 points]

Many loops repeatedly access array locations as they result from the trivial translation of high-level programming constructs. The example below, used to implement what is commonly known as a “stencil” computation moves across a 1D “window” of the ‘a’ array locations updating a second array ‘b’.

```c
for(i=1; i < N; i++)
    b[i] = (a[i-1] + a[i])2.0;
```

In this exercise you will apply a specific transformation, called scalar replacement as illustrated in the figure below. Here the array references are converted to scalar references by using two additional temporary variables and “rotating” the values across the temporaries.

```c
r0 = a[0];
for(i=1; i < N; i++)
    r1 = a[i];
    b[i] = (r0 + r1)/2.0;
    r0 = r1;
```

This great advantage of this “code scheme” is that each array location for the array ‘a’ is fetched only once. Also, you can “easily” extend this transformation to other loop nests such as 2D structures where you manipulate a 2D array as is the case of many image processing kernels.

Questions:

a. [05 points] When is it legal to apply this transformation?
b. [10 points] Suggest a simple code transformation algorithm possibly using loop peeling as an auxiliary transformation.
c. [10 points] Apply your algorithm to the code below.

```c
for(i=1; i < (N-1); i++)
    b[i] = (a[i-1] + a[i] + a[i+1])/3.0;
```
Problem 3: Iterative Data Flow Analysis [35 points]

In this problem you will develop and show the application of the Live Variable analysis problem to the code of a given procedure. The live variable analysis problem seeks to determine for each scalar variable (or temporary register) if its contents is live. This has the fundamental application in the context of register allocation since if a given variable is no longer live at a given point then there is no need to keep it in register any longer.

Formally, a variable \( v \) is live at a execution point \( p \) if either its value is used at \( p \) or there exists an execution path from \( p \) to \( q \) along which the value of \( v \) is not written and is used in \( q \).

a. [10 points] Formalize the live variable problem as an iterative data-flow analysis problem showing the equations for the gen and kill abstractions as well as the initialization of the values for each basic block and statement. In this section you need to determine if this is a forward or backward data flow problem.

b. [15 points] Apply your data flow problem formalization to the procedure code shown below (do not apply any transformations) showing the final result of the IN and OUT set of live variables for each basic block in the code.

c. [10 points] Consider that you only have 1 register to implement the code in the body of the loop. Which set of live variables would you keep in registers and why?