CS4XX INTRODUTION TO COMPILER THEORY

Week 14

Reading:

Chapter 10 from Principles of Compiler Design, Alfred V. Aho & Jeffrey D. Ullman

Objectives:

1. To learn the concepts of Code generation

Concepts:

1. Code optimization Introduction----------1 hour
2. Code optimization concepts---------- 2 hours

Outlines:

1. Introduction
2. The principal sources of optimization
3. Optimization of basic blocks
4. Loops in Flow Graphs

CS 4XX Week 14 – Lecture Notes

1. Code Optimization

a. Introduction

- Criteria for Code-Improving Transformations – The steps that can be followed to improve code transformations.
- Getting Better Performance – Methods that can be used to improve the performance of the system.
- An Organization for an Optimizing Compiler – The levels at which a program can be improved, to optimize the compiler performance.

b. The Principal Sources of Optimization

- Function-Preserving Transformations – Methodologies for a compiler to improve the performance of a program without changing its functionality.
- Common Subexpressions – How recurring expressions can be kept out of the compilation process to improve performance.
- Copy Propagation – The use of an assigned expression to reduce the compiler time in a larger expression.
- Dead-Code Elimination - Eliminating variables that are no longer in use.
- Loop Optimizations – Optimizing inner loops to improve the performance.
- Code Motion – Separating repeating expressions and placing them outside the loop to save execution time.
- Induction Variables and Reduction in Strength – An quick sort example of an inside-out loop and usage of induction variables.

C. Optimization of basic blocks

a. Structure preserving Transformation

1. In order to code improve transformation for basic blocks structure preserving transformation is included such as sub expression elimination and dead-lock elimination

b. Use of Algebraic Identities

1. Replacing more expensive more expensive operator bya cheaper one.

c. Language specification

1. It should be made carefully, to determine what rearrangement of computations is permitted.

D. Loops in Flow Graph

a. Dominators

1. Dominator Tree -- Useful way of presenting dominators information is in a tree, called Dominator tree. In which the initial node is always root, and each node dominates only its descendents in a tree.
2. Unique immediate dominator --- The existence of dominator trees follows from a property of dominators; each node n has a unique immediate dominator m that is the last dominator of n on any path from the initial node to n.

b. Natural loops

3. Find out all the loops in a flow graph.
4. Search edges in the flow graph.
5. Essential property of Natural loop

6. Loop must have a single entry point, called the header. This entry point dominates all nodes in the loop, or it would not be the sole entry to the loop.
7. There must be at least one way to iterate the loop, i.e. at least one path back to the header.

c. Inner Loops and Pre-Header

1. One that contains no other loops --- unless two loops have the same header, they are either disjointed or one is entirely contained the other loop.
2. Neglecting loops with the same header for the moment, we have natural notion of inner loop which contains no other loops.
3. Basic Requirement of Pre-Header --- Several transformations require us to move statement before the header, we therefore begin treatment of a loop L by creating a new block, called the PREHEADER.

d. Reducible flow graphs

1. When to reduce Flow Graph --- Exclusive use of structured flow of control statements whose flow graphs are always reducible.
   a. if-then-else,
   b. while-do,
   c. continue,
   d. break statements
2. Only entry to a loop is through its header --- It is an important property of reducible flow graphs, namely that there are no jumps into the middle of loops from outside.

3. Forward Edges and Back Edges --- A flow graph is reducible if and only if we can partition the edges in to two disjoints groups, often called the forward edges and back edge.