Michgan Tech

The implementation of SVN and LVN

Project II

Writen by: liang YAN

Computer Science

Last modified: February 2014

Project Requirement

1. We will continue basing our project on the Low Level Virtual Machine (LLVM) compiler infrastructure. In this project you will implement and/or compare three value numbering techniques, local value numbering (LVN), super local value numbering (SVN) and dominator-tree-based global value numbering (GVN).

2. GVN has already been implemented in LLVM (see (llvm)/lib/Transforms/Scalar/GVN.cpp). Note that GVN subsumes SVN, which, in turn, subsumes LVN. You need to update the GVN code so that it can be downgraded to perform LVN or SVN only. To implement super local value numbering, CS5130 students need to implement an analysis pass that identifies extended basic blocks. CS4130 students are not required to implement this pass. However, you will need an algorithm to determine if basic block A is an ancestor of basic block B in an EBB.

3. You will need to introduce two new command line options for opt which suggest the optimizer to perform LVN or SVN instead of GVN. I will use command line option -std-compile-opts to trigger standard 1compiler optimizations which include global value numbering. You should add the following two options to opt:
   1. -lvn: perform local value numbering only.
   2. -svn: perform super local value numbering only. CS5130 students need to output EBBs for each function if this option is enabled. Most of your work is to understand GVN.cpp and make changes to integrate local and super local value numbering into it.

4. Your code must work on the Rekhi lab machines. It is fine to use your own machine for development. But you must test your code in the lab machines.
   Download and install test-suite-3.4 from the LLVM website. We will use test-suite-3.4/SingleSource/Benchmarks/Stanford as the testing benchmark suite. Your optimizations should pass all the benchmarks in this suite. You also need to collect the total compile time, the compile time of LVN, SVN and GVN, the total execution time, and the number of instructions deleted by value numbering.

5. Extended Basic Block (CS5130 students only)
   Dump all the extended basic blocks in each function led by the function name followed by a colon. Each EBB should be output as a pre-order sequence of the basic blocks in it enclosed by a pair of curly brackets and each basic block can be represented by its name.

6. Submission
   Tar all the source files you have changed including their directories into a tar ball,

project2.tar, and submit it through Canvas. Write a report that summarizes your implementation and analyzes your experimental results for the Stanford benchmark suite.

CS5130 students should also attach your EBB outputs and name them after the original input file names with .ebb extension. For example, the EBB output for Bubblesort.c should be stored in file Bubblesort.ebb.

0.1 Introduction

GVN,this class uses global value numbering to eliminate fully redundant instructions. It also performs simple dead load elimination. To implent global value numbering, GVN create the value table class to hold the mapping between values and value numbers, it also uses a structure LeaderTable which mapping from value numbers to lists of Value*'s that have the same value number. Basically, GVN get the domaint tree from the runonfunction, and do the valuing work based on the this domainator tree. When meet an instruction, first check it could be emliminated or not(it is not the valuing job, but only for convient),then put the instruction into VN(value nuber table), aslo add to LeaderTable, meanwhile, it will check the current LeaderTableEntry and check it with all basic blocks which domainted it, if found one, means, current one could be eliminated. so the number is mapping with BB and instruction.

0.2 Implementation

0.2.1 SVN

UNlike the GVN, I did not care about the load emiliatin so much, what I do is use a EBBForest class instead of domainator tree, iterating EBBTree sepeatly, and insert value to vn and LeaderTable, but when doing findleader job, I only check the basic block in LeaderTable which could extend it.

0.2.1.1 LVN

I use the original BB from function, and only focused on locally

0.2.2 Command-line

There are two mechinism of argument passing in LLVM, one is use the parament directly, like "Release+Asserts/bin/opt -svn test.bc ¿¿ /dev/nul ", in this way, I also need to put this command to std-compiler-opts, There is another easy way, just use it as a condition variable of GVN ,use it like this "Release+Asserts/bin/opt -gvn –enable-lvn=true test.bc ¿¿ /dev/nul " , then add static cl::opt¡bool¿ EnableLVN("enable-lvn",cl::init(false), cl::Hidden);

0.2.3 EBB

I designed a EBB pass here,

0.3 Result anlysis

The result look like below:

Name	GVN	SVN	LVN	GVN	SVN	LVN
Bubblesort	4	0	8823859	0.0004	8823859	0.810975
FloatMM	7	0	8819563	0.0005	8823859	0.810975
IntMM	7	46680967	8823859	0.0002	8823859	0.810975
Oscar	5	46680967	8819563	0.0007	8823859	0.810975
Perm	0	46680967	8823859	0.0002	8823859	0.810975
Puzzle	0	46680967	8819563	0.0014	8823859	0.810975
Queens	0	46680967	8823859	0.0004	8823859	0.810975
Quicksort	12	46680967	8819563	0.0003	8823859	0.810975
RealMM	7	46680967	8819563	0.0003	8823859	0.810975
Towers	1	46680967	8823859	0.0005	8823859	0.810975
Treesort	6	46680967	8819563	0.0008	8823859	0.810975

0.4 The difficulties

1. To understand how these predictiors work, this is also the purpose of this project. The TA gave me a lot of help for this problem, also I reviewed the class record, finally I figured it out, it felt good.

2. I also need to

0.5 Conclution