Michgan Tech

The implementation of SVN and LVN

Project II

Writen by: liang YAN

Computer Science

Last modified: February 2014

Project Requirement

1. In this project, 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.

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.
   3. output EBBs for each function if this option is enabled.

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
   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, 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

This class uses global value numbering,GVN, to eliminate fully redundant instructions. It also performs simple dead load elimination together.
To implent global value numbering, GVN create the value table class to hold the mapping relation between values and value numbers, it also uses a structure LeaderTable which mapping from value numbers to lists of Value*'s which have same value number. Basically, GVN does the valuing number work based on a domainator tree.It uses a dependency method to get the domaint tree , and then use a DFS iteration go through the whole dominator tree. This happens in the function "iterateOnFunction".Then it is instruction processing. When meet an instruction, first check it could be emliminated or not(it is not the valuing job, but only for convient, like merge block, simplicathin instruction), then put the instruction into VN(value nuber table), aslo add to LeaderTable, this will generate two numbers. According the two numbers, it will compare the current LeaderTableEntry with the other basic block numbers in the dominator tree and dominates it.If found one, means, current one could be eliminated. Since a function has one dominator tree, we only do clean job once.

0.2 Implementation

Based on the analysis of GVN, we could find that, SVN and LVN could be easily implented by offering the class a special block and limited the checing range to this special blocks. For GVN, it is a dominator tree,For SVN it is an EBB, and For LVN , it is only one single basic block.

0.2.1 SVN

UNlike the GVN, I did not care about the load emiliatin so much, what I do is use an EBBForest structure instead of domainator tree, iterating EBBTree sepeatly, and insert value to vn and LeaderTable.When doing findleader job, I only compare the basic block with those blocks in LeaderTable and could extend it during the EBBTree.

0.2.1.1 LVN

Same idea about SVN Implementation, I processed one block one time.

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”, in this way, I need to put this command to std-compiler-opts,and solve some problems about the dependency relationship. So far, I only find the flow of a argument, there are still some work on how let a svn class use gvn class well.
There is another way,easy way, just use it as a condition variable of GVN ,like ”Release+Asserts/bin/opt -gvn –enable-lvn=true test.bc” , then add static cl::opt¡bool¿ EnableLVN(”enable-lvn”,cl::init(false), cl::Hidden); to front of the file, then we could use EnableLVN as a global bool type variable in this file.

0.2.3 EBB

I designed a EBB pass in this way:
First, construct an EBB tree class to implent an Extend basic block, it has a root to save the entry block, it also has a vector structure to save all block. I tried to create a new map structure to map a basic block ant its ancestors, but because of the time, I did not finish yet.This class also provides some functions like verify the ancestor.
Secondly, I counstruct an EBBForest class to save all ebbs in a function. Considering there would be multiple functions in file, I create an map structure to connect the function name and ebb forest.
The main function Implemented in ”runOnFunction”, I also provide a function to save all ebbs in a file.

0.3 Result anlysis

The result look like below:

Delete NO	GVN	SVN	LVN
Bubblesort	7	7	7
FloatMM	5	6	6
IntMM	6	6	6
Oscar	7	7	6
Perm	3	3	2
Puzzle	23	17	27
Queens	3	3	0
Quicksort	14	13	12
RealMM	6	6	6
Towers	5	5	59
Treesort	13	13	13

Time	GVN	SVN	LVN
Bubblesort	0.0002	0.0002	0.0002
FloatMM	0.0002	0.0002	0.0002
IntMM	0.0002	0.0002	0.0001
Oscar	0.0004	0.0004	0.0003
Perm	0.002	0.0003	0.0002
Puzzle	0.001	0.0009	0.001
Queens	0.0002	0.0002	0.0002
Quicksort	0.0002	0.0002	0.0002
RealMM	0.0001	0.0001	0.0001
Towers	0.0004	0.0004	0.0005
Treesort	0.0003	0.0003	0.0003

Compile T	GVN	SVN	LVN
Bubblesort	0.004	0.0039	0.004
FloatMM	0.0048	0.0047	0.00473
IntMM	0.0054	0.005	0.0046
Oscar	0.0084	0.0084	0.0087
Perm	0.0031	0.0032	0.003
Puzzle	0.0286	0.0279	0.0276
Queens	0.0051	0.005	0.005
Quicksort	0.0043	0.0043	0.0038
RealMM	0.0044	0.0045	0.0045
Towers	0.006	0.0058	0.0060
Treesort	0.0052	0.0051	0.0051

Execute	GVN	SVN	LVN
Bubblesort	0.005	0.005	0.005
FloatMM	0.001	0.004	0.006
IntMM	0.005	0.005	0.004
Oscar	0.009	0.009	0.011
Perm	0.0002	0.004	0.003
Puzzle	0.03	0.028	0.025
Queens	0.006	0.004	0.004
Quicksort	0.003	0.004	0.003
RealMM	0.003	0.004	0.004
Towers	0.004	0.006	0.005
Treesort	0.006	0.004	0.003

0.4 Conclution

According to the data below, we could find that:
1. Comiler time takes a big proportion of the total time, nealy equal to execute time, even bigger sometimes.
2. From the exectute time, we find that gvn < svn<lvn, which means a good optimization does improve execute effiency of program
3. Each optimization time is less than 10 percents of total compiler time, so optimization is necessary.
4. There is a relationship between code numbers and execute time, usually less code less time.
5. A good opitimization improves effiency indeed, but it will improve the compile time at the same time. So a good algorithm is important even for opitimization Implementation.