Lab 2 Report

Yuanqing Zhu

yz120

To Submit Your Code:

Create a tar file with your source code, a makefile, and a README file that explains how to build your lab and how to run it. Send that tarfile, via email, to `keith@rice.edu`.

To Submit Your Report:

Answer the questions below. You may edit this document directly and submit via email — rename the file to your netid so that your filename does not clash with the filenames used by other students.

You may submit this document on paper by answering the questions and turning it in to my office, DH 2065. You can slide it under the door if I am not there.

Be sure to write down your name and your netid.

Quantitative Results

For each of the benchmark codes in `~comp506/students/lab2/TestCodes` on the **CLEAR** systems, complete the following chart.

Bench-mark	Input	Simulated Cycle Counts			
		Original	Value Numbering	Loop Unrolling	Both
bsort	bsort2.d	42,359	39,041	42,215	38,897
qsort	qsort2.d	15,505	14,518	15,361	14,374
fib	20	1,753	1,753	1,627	1,627
mmult	50	8,425,269	7,372,669	8,172,819	6,396,619(-u-v) 7,120,219(-v-u)
sumred	sumred1.d	4,272	4,072	4,092	3,412(-u-v) 3,892(-v-u)
algred	50	2,520,408	2,270,408	2,272,908	1,482,908(-u-v) 2,022,908(-v-u)
oneloop	50	3,510,218	3,260,218	3,510,218	3,260,218

The code produced by your lab should produce the same answer as the original code— that is, both the original and the optimized code should print the same numbers in the output. (Of course, the simulator report on instructions executed and cycles executed should be different.)

Experience

(1) Briefly discuss your experience building the optimizer. What things were easy to do? What things were hard?

This time, the work is divided into 4 pieces. Parse the input, value numbering, loop unrolling and output. I just chose to use the fstream to read in and out the .i files. This part is repetitive because of the way I chose. I simply analysis all lines in the file, and capsulate each line into a well-designed instruction struct. Also, I do one more step to construct two vectors to store all the start and end line numbers for all blocks, which would be used in both two functions. These parts are not hard. The crucial point is to design the struct and analysis each line, and then do similar work for each line. Also, output is easy.

Inside value numbering part, I simply use several maps to store the relation between each operand and the value number. This part is kind of tricky when I need to use 2 or more maps at the same time. I need much time to make some drafts and think it over. The value number is changing and the relationships in the map is changing. Kind of intricate.

Inside loop unrolling part, firstly, I made some mistakes. I simple loop statements (a,b,c) in an inner loop to (aaaa,bbbb,cccc), which is not identical with the slices. After changing to abc for 4 times, more problems appear. First, some input are not the times of 4. Then I do some computation at first when it comes to a loop. Like 9 (binary: 1 0 0 1), I right shift it by 2 digits and then change it back and get the biggest number smaller than it with the times of 4. (1 0 0 0) This means I need to do only 8/4 = 2 times of by-4-loop and 9-8=1 time of by-1-loop. Then I create several new blocks to do new by-4 loops and the changes of branches. Also, a helper function for computing new variable and new label is implemented. This part can be hard because of many conditions need to be considered. Many new instruction structs and blocks are created and some statements in the original test case would be reduced and changed, which makes this part of problem kind of confused.

Experience *(continued)*

(2) Did you implement any extensions to value numbering?
I find all the redundant part in computing and change them to i2i at first, which would not reduce the cycles.
Also, when browsing the internet, I found an idea from others, that we can use lshiftI to substitute the multI, if the left operand is a power of two. This should be much faster, as to the experience when dealing with the algorithm problems. Bit manipulations are quite faster. I implemented this part as well.
For example, in the bsort code, without substituting multI with lshiftI, the cycle number after value numbering is 42359(didn't change), while it can be reduced to 39041 after using this extension.

Experience *(continued)*

(3) How did you decide which loops to unroll.
I parse all instructions every time before executing value numbering or loop unrolling. This helps me to keep the correct start and end position for each block. Then, for each block with a “br” or “cbr” as an end, I figured out whether it can loop back to its own block, which means the right operand in “br” or “cbr” should be identical to the label of the block. If so, it is an inner loop.
Next, I rewrite this block into: 1. A computation block, to compute how many times do I need to execute the by-4 loops and by-1 loops. 2. A new block of by-4 loop. 3. An intermediate block for jumping from by-4 loop to by-1 loop. 4. The original by-1 loop.

Experience *(continued)*

(4) What, if anything, would you do differently if you could start over on this project?
I may use flex and bison to deal with the input file, because the method I used right now is repetitive. Also, flex and bison files exist in Sim, but need time to understand and make some modifications. This time, I already started to write some codes, then I found the existing codes. After consideration, I decided to move forward with my own codes.

Experience *(continued)*

(5) If you could change the assignment, what would you change?
Not a big deal, but I may add a basic parse function inside the initial codes. Or just part of them. Because analysis each line or each operator is boring and repetitive. Also, using the flex and bison can be an alternative, but for this problem, I failed to use the existing parsing part in Sim and to integrate in my code. More context free grammar texts should be provided for better understanding the parse part codes.