**Lab 3 Report**

*Nai-Fan Chen*

*nc41*

**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 comp412code@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, where O1 and O2 are the two optimizations that you built.

Note: After running the program, I will generate output.i to store the result. It will stdout as well.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | **Simulated Cycle Counts** | | | | |
| **Bench-mark** | **Input** | **Original** | | **V** | **U** | **V then U** | **U then V** |
| bsort | bsort2.d | 42,359 | | 39,041 | 39,041 | 39041 | 39041 |
| qsort | qsort2.d | 15,505 | | 14,362 | 14,362 | 14362 | 14362 |
| Fib | 20 | 1,753 | | 1,753 | 1,537 | 1537 | 1537 |
| mmult | 50 | 8,425,269 | | 7,902,669 | 7,902,669 | 7902669 | 7902669 |
| sumred | sumred1.d | 4,272 | | 4,072 | 4,072 | 4072 | 4072 |
| algred | 50 | 2,520,408 | | 2,270,408 | 2,260,408 | 2010408 | 1530408 |
| oneloop | 50 | 3,510,218 | | 3,260,218 | 3,510,218 | 3260218 | 3260218 |

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?**

The easy part is parsing the iloc code into my optimizer. I directly use the source code provided by professor in simulator. I can get all the data from there, and then I build linked list to store them into my own data structure.

The hard part is to determine where to unroll the loop because there are nested loops and some specific operations will cause unrolling to produce wrong answer like br, cbr. Furthermore, when unrolling, I need to design new labels to put the unrolling code. It’s difficult to design a stable function to update all the related branches and jump back to original place.

**Experience** *(continued)*

1. Which optimizations did you implement?

What problems arose in your implementation?

Did you extend them in any interesting way beyond the basic algorithms?

1. I implemented Value Numbering and Loop Unrolling.
2. Value numbering needs to assign every variable and every const a unique value number. At first, I didn’t do it and I did not use value numbers in the expression but the original name, so it will eliminate wrong expressions.

Loop unrolling, I did not detect nested loop at first, and I cannot jump back to the original label because there are another cbr inside the outer loop. Thus, I change the target loop to the deepest one only.

1. I append loop unrolling codes at the last of iloc code and, since there is a halt at the end of the original code, the codes will not reach except br or cbr to jump there. I believe this is very beneficial because I do not need insert the code between original code. That is, it is easier to debug and implementation.

I turn the multiply and divide into lshiftI and rshiftI. I found that this is very efficient if there are many divide/multiply operations.

**Experience** *(continued)*

1. If you implemented loop unrolling, how did you decide which loops to unroll?

I assume I have known this is a loop. I will check whether there are other loops inside current loop. If there are, I would get inside to the deepest loop and unroll it. Since the nested loop is very hard to make it correct, I only unroll the single loop. Furthermore, if the loop does not contain arithmetic, I would not unroll it either because some other opcodes would be difficult to maintain.

**Experience** *(continued)*

1. What, if anything, would you do differently if you could start over on this project?

I will use C++ instead because I do not need to build the linked list or Hashmap. I spend lots of time on it. Moreover, I will add the target Label pointer in my basic blocks to implement the ebb. I would also build a better function structure to make my code more concise.

For the unrolling part, I would like to check the length of the loop since long codes in loop do not have much improvement. However, short one is very beneficial because the loop’s overhead is very heavy. Thus, if the loop is short, the partition of the loop’s content is relatively small, so the overhead is relatively large. That’s why I would like to focus on smaller loop content.