Programming Homework 4 for CA

Due Dec 18th

You should do this assignment on your own. No late assignments!

If you have any problems, please contact the teaching assistant in time.

In this assignment you will learn about two different methods of exploiting instruction level parallelism: “branch prediction” and “predication”. You will learn and experiment how they differ from each other. You can make suitable changes to the code to finish the assignment.

Step 1

Consider the following piece of code:

```
if (x < y)
   x = y;
```

There at least two ways in which we can generate the assembly code for this.

1.Using branches:

```
  compare x, y
  jump if not less L
  move x, y
L:
```

2.Using conditional move:

```
  compare x, y
  conditionally move x to y.
```

Each of the two versions has its own advantages. Which version should one prefer? We will try some examples about them.

Here are some posts on cmov from Linus Torvalds, the creator and maintainer of the Linux operating system. Linus has provided a short piece of C code for measuring the performance of branches and conditional moves. Run the code on your x86 processor and report the running time for the two versions choose() function. You should run each version at least 10 times. Report both the average execution time and the standard deviation in the run times. If you see too much variation in the run times, run for more iterations. This should typically stabilize the performance.

Then simulate the same two versions with gem5 using the out-of-order/O3 processor. Lower the number of iterations to 1,000,000 since 100,000,000 is lot of iterations for gem5. Again report which option performs the best. Also report the total number of branches predicted and the number of branches predicted incorrectly.

Step 2

A paper on branch avoiding algorithms was published at SPAA 2015. The authors suggest that graph algorithms that avoid branches may perform better than algorithms that use branches. Let's try to verify this claim.

The paper provides two versions of an algorithm for computing the connected components in an undirected graph. The first version uses branching and the second one uses conditional moves. The corresponding C++ file is connected-components.cpp . And three graphs: small.txt, medium.txt and large.txt. Read the C++ source to understand how to supply options to the executable.

Run both the versions (with branches and with cmov) on an x86 processor and report the run time performance for the provided input files. Do this exercise only for the large graph. Report data as asked in step1.

Run both the versions with gem5, report the performance of the two versions for the annotated portion of the code, the number of predicted branches, percent of incorrectly predicted branches. You need to do this only for small and medium graphs, not for the large one. Report data as asked in step1.

What to Hand In

Turn in your assignment on Canvas

- All files below should be attached as a zip file. The submission name should contain the name and ID numbers of the student, for example: zhangsan_921104681912_HW4.zip.
- A report for all the questions in step1 and step2 with necessary notes. And when you report time, attaching your processor information is best. Recommended submission formats are doc, pdf and md.
- choose.cpp if you have change in step1, connected-components.cpp if you have change in step2, and graph.txt if you create a graph in step2.

Some Hints

- If you are using a non-X86 processor, please indicate it in your report.
- If you think the step1's posts confusing, here is the code to run.

```c
#include <stdio.h>

/* How many iterations? */
#define ITERATIONS (100000000)

/* Which bit of the counter to test? */
#define BIT 1

#ifdef CMOV

#define choose(i, a, b) ({                      \
        unsigned long result;                   \
        asm("testl %1,%2 ; cmovne %3,%0"        \
                :"=r" (result)                  \
                :"i" (BIT),                     \
                 "g" (i),                       \
                 "rm" (a),                      \
                 "0" (b));                      \
        result; })

#else

#define choose(i, a, b) ({                      \
        unsigned long result;                   \
        asm("testl %1,%2 ; je 1f ; mov %3,%0\n1:"       \
                :"=r" (result)                  \
                :"i" (BIT),                     \
                 "g" (i),                       \
                 "g" (a),                       \
                 "0" (b));                      \
        result; })

#endif

int main(int argc, char **argv)
{
        int i;
        unsigned long sum = 0;

        for (i = 0; i < ITERATIONS; i++) {
                unsigned long a = 5, b = 7;
                sum += choose(i, a, b);
        }
        printf("%lu\n", sum);
        return 0;
}

```

- Since this assignment was proposed, computers have grown by leaps and bounds, so the difference between the two versions may not be particularly noticeable. It maybe only a few milliseconds. You can try to improve the time precision or change the number of iterations to get a more significant difference. Also, you can create a large enough graph in step2.
- In the step2, the getline() function in ReadInputFile() can only work on linux. You can rewrite the ReadInputFile() function to avoid errors.
- In the origin homework, the author says he couldn't get CMOV instruction working with inline assembly, so he providied a GCC generated-assembly code. But I successfully ran both versions. If you have problems with CMOV version, you can compile the connected-components.s I provide in this zip file.