DUE DATE: **WED, SEPT 12 2017 BEFORE 11:59 PM**

zCSE-443/543: High Performance Computing

Exercise #4

Max Points: 20

You should save/rename this document using the naming convention MUid.docx (example: `raodm.docx`).

Objective: The objective of this exercise is to:

- Understand creating and running microbenchmarks
- Learn process of submitting jobs on Red Hawk
- Determine if a `switch` statement can improve performance when compared to a semantically equivalent `if-else` construct using a micro-benchmark.

Fill in answers to all of the questions. For almost all the questions you can simply copy-paste appropriate text from the shell/output window into this document. You may discuss the questions with your instructor.

Name: **Sam Horn**

Warnings from C++ compilers should be taken seriously because in many cases warnings indicate a potential runtime problem with your program. Consequently, all grading, particularly for homework, will require that programs compile without warnings or style issues. Points will be deducted for each warning generated by the compiler and style issues programs.

Background

C++ provides two different approaches to implement control flow either with a series of `if-else` statements or using a `switch` statement. Recollect that in C++, the `switch` statement can be used only with primitive data types (such as `int`), which gives the compiler an opportunity to optimize the `switch` statement (if possible). Often there are questions and discussions (search `stackoverflow.com` if you are interested) about the performance difference between `if` vs. `switch`.

Scientific question:

Is there a performance difference between using a series of `if-else` statements versus using a `switch` statement?

Hypothesis:

When a `switch` statement is used for a contiguous range of `int` values (say 0…9) the compiler can better optimize the `switch` statement when compared to a semantically equivalent series of `if-else` statements.

Design of Microbenchmark

In order to verify the above hypothesis (i.e., accept or not accept the hypothesis) the following operations needs to be performed:

1. A suitable Microbenchmark needs to be developed with semantically equivalent `if-else` statements and `switch` statements that operate on a contiguous range of integer values.
2. Run the benchmark using the 2 different constructs to measure runtime
3. Compare runtime with different optimization levels to see if a difference in runtime is observed.
4. Using the runtime data conclusions need to be drawn if one construct is better than the other in the given scenario.

Review a Microbenchmark

Estimated time: 15 minutes

In this exercise you are given a starter code that already implements the microbenchmark. Do not modify the benchmark but carefully review it to ensure you understand the benchmark. Recollect that a lot of care and attention is necessary to develop good, meaningful benchmarks.:

Procedure

1. Create a remote C++ `NetBeans` Project named `Exercise4` on the `redhawk.hpc.miamioh.edu`.
2. Copy the supplied starter code in `Exercise4.cpp` to the corresponding file in your `NetBeans` project.
3. Study the starter code and note the following key aspects
    a. The `useSwitch` method implements the logic using switch statements.
    b. The `useIf` method implements the same functionality using a series of `if-else` statements.
    c. As yourself these important (exam) questions --
        i. why is `switchTest` and `ifTest` methods being called so many times in a loop?
        ii. Why can't we directly call `useSwitch` and `useIf` methods and eliminate intermediate `switchTest` and `ifTest` methods, thereby possibly making the benchmark simpler?
4. Validate your implementation using the following command-line for a short run (it should run in < 1 second):

```
$ ./Exercise4 switch 5
Sum = 8845740
$ ./Exercise4 if 5
Sum = 8845740
```

Needless to add, both versions (`switch` and `if-else`) should produce the same results.

Apparatus (experimental platform)

The apparatus used for the experiments documented in this report were conducted on the following platform (replace commands shown in grey with correct information obtained using the command in grey):

Component	Details
CPU Model	Intel ® Xeon® CPU X5550 @ 2.67GHz
CPU/Core Speed	2661.000 MHz
Main Memory (RAM) size	24591648 kB
Operating system used	Linux mualhpcp01.hpc.muohio.edu 2.6.32-642.el6.x86_64 #1 SMP Tue May 10 17:27:01 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
Interconnect type & speed (if applicable)	n/a
Was machine dedicated to task (`yes/no`)	Yes (but with minor load from general purpose software and system processes)
Name and version of C++ compiler (if used)	G++ (GCC) 4.9.2
Name and version of Java compiler (if used)	n/a
Name and version of other non-standard software tools & components (if used)	`callgrind` profiler and `kcachegrind` (GUI viewer)

Experiments and Observations

Estimated time: 25 minutes

Now you will need to compile and run your microbenchmark using PBS jobs using the following procedure:

1. Copy the supplied PBS job script `ex4.bash` to your `NetBeans` project. Briefly review the supplied script to understand its operations by recording the following information:
   How much time is requested for the job: 10 minutes

   How many cores are requested for the job: 1

   How much memory is requested for the job: 128MB

2. Next compile your microbenchmark from a terminal window using the following standard command-line (you should have the following command memorized by now):

```
$ g++ -g -Wall -std=c++14 -O2 Exercise4.cpp -o Exercise4
```

3. In your `ex4.bash` ensure your job name and the benchmark command-line are consistently setup for the configuration you are planning to benchmark. For this setting you don't need to specify number of iterations as it is assumed to be default 2 billion.

4. Next submit a PBS job via the following command line:
```
$ qusb ex4.bash
```

5. Wait for your job to complete -- that is the last character should be a '`C`'. Completed jobs are unlisted 10 minutes after their completion. You may monitor the status of your job using the following command-line:
```
$ qstat -u $USER
```

6. From the output file from the job (you can open the file via `NetBeans`, use 3rd icon/tool in `remote tools` sub-toolbar) record the necessary statistics in the table(s) below.

7. Now repeat steps 2-6 for `switch` and `if-else` versions using both `-O2` and `-O3` optimizations levels (total of 4 combinations). Pay attention to flags and settings in `ex4.bash` to ensure you don't mix/jumble settings.

Observations:

Record the timing information collated from your experiments conducted using your micro-benchmark in the tables below:

Using if-else statements	Elapsed Time (sec) measured using `/usr/bin/time` command	
	Opt: -O2	**Opt: -O3**
Observation #1	50.28	25.09
Observation #2	50.28	24.90
Observation #3	50.47	25.59
Averages (of 3 runs)	**50.34**	**25.19**

Using switch statements	Elapsed Time (sec) measured using `/usr/bin/time` command	
	Opt: -O2	**Opt: -O3**
Observation #1	18.41	14.47
Observation #2	18.18	14.44
Observation #3	18.18	14.44
Averages (of 3 runs)	**18.25667**	**14.45**

Results and Conclusions

Using the above data develop a report (no more than 10 sentences) discussing the performance aspects and conclude if you accept or reject the hypothesis. Discuss percentage and speedup in runtime with different compiler optimization levels. You may also mention data that you may not have recorded (as in peak memory usage) as needed. Indicate what suggestion you would provide to a programmer based on your experiment.

The collected data from the micro-benchmark tests shows that switch statements outperforms the if-else statements in both optimizations. Each test has a minimum of 98% CPU usage on 1 core suggesting these programs were not starving the CPU. The -O3 optimization outperformed the -O2 optimization for both statements, increasing the switch 33.04% and the if statement by 98.25%.

This micro-benchmark tests shows that the CPU are able to optimize switch statements significantly more than switch statements. That being said, the switch statement is this situation is easier to comprehend from a programmer's perspective and preferred to an if-else when consider over many branching paths.

Submit files to Canvas

Upload the following files to Canvas:

1. Upload this MS-Word document (duly filled with the necessary information) saved as PDF using the naming convention **`MUid.pdf`**.