**CSE-443/543: High Performance Computing**

**Exercise #8**

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 gain some familiarity with:   * Working with profilers, specifically perf. * Understanding control hazards and branch prediction in action.   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. |

|  |  |
| --- | --- |
|  | **Maciej Wozniak** |

# Background

Detecting regions with performance issues in program is a challenging task, particularly in large software systems. Profilers are special software programs that instrument, monitor, record, and report performance profile of a given program. The performance profile provides detailed runtime statistics about the program. The profile information is used to identify performance bottlenecks or "hot spots" in programs.

**Note**: Performance profiles only enable identification of "hot spots". They do not provide information on how to fix the performance bottleneck. Fixing the performance bottleneck will require appropriately reworking the program using different algorithms, data structures, or instructions.

# Part #1: Submit job to run with perf profiler

*Estimated time: 15 minutes*

**Exercise**

1. Log into OSC’s OnDemand portal via <https://ondemand.osc.edu/>. Login with your OSC id and password.
2. Startup a VS-Code server and connect to VS-Code. Ensure you switch to your workspace. Your VS-Code window should appear as shown in the adjacent screenshot.
3. Next, create a new VS-Code project in the following manner:
   1. Start a new terminal in VS-Code
   2. In the VS-Code terminal use the following commands:

|  |
| --- |
| $ # First change to your workspace directory  **$ cd ~/cse443**  $ # Use ls to check if workspace.code-workspace file is in pwd  $ # Next copy the basic template for a C++ project  **$ cp -r /fs/ess/PMIU0184/cse443/templates/basic exercise8**  $ # Copy the starter code for this exercise  **$ cp /fs/ess/PMIU0184/cse443/exercises/exercise8/\* exercise8** |

1. In this exercise we will reuse the solution for Exercise #7 which compared the performance of if vs. switch statements.
2. Review the supplied SLURM script for this project. Note how the script compiles and runs the program.
3. Submit your SLURM job as

|  |
| --- |
| $ sbatch Exercise8\_slurm.sh |

The above job will be queued and then take several seconds to start and run. Once the job ends, you will see an output text file in the format slurm-*<JobID>*.out (e.g., slurm-12345.out) with the results.

|  |  |
| --- | --- |
| ☞ | **Note**: While you are waiting for your job to run complete the next part of this exercise. |

# Part #2: Review profiler background

*Estimated time: 10 minutes*

1. State one advantage and one disadvantage of hardware profilers?

|  |
| --- |
| + more accurate; - expensive |

1. What is the difference between a hardware profiler and a hardware-assisted profiler?

|  |
| --- |
| **Hardware profiler** : purely specialized hardware for profiling purposes, **Hardware assisted profiler**  uses partially hardware partially software for efficient profiling, has to use specialized operating system |

1. Briefly describe 3 ways in which software profilers instrument a program/code to gather profile information.

|  |
| --- |
| 1. **Manual**: User would defined (by writing some commands) which lines of code has to be profiled 2. **Compiler assisted**: compiler insert the code in order to make use profiling 3. **Runtime instrumentation**: binary exe file is being modified directly just before running |

# Part #3: Record & analyze profiler information

*Estimated time: 10 minutes*

Once your job completes, record the following information from output of the job file. Record the average of the statistics from 3 runs of Linux perf:

|  |  |  |
| --- | --- | --- |
| **perf statistics (average of 3 runs)** | **Using switch** | **Using if** |
| Time elapsed (seconds) | 6.65 | 6.51 |
| instructions | 45,049,349,536 | 82,798,860,843 |
| CPI (insns per cycle) | 1.87 | 3.51 |
| branches | 2,446,278,328 | 15,446,161,068 |
| branch-misses (i.e., cases where CPU prediction was incorrect) | 144,5836 | 128,251 |

From the above table, note how the instructions executed for the if-case is much higher. Then how is it running almost at the same speed (if not slightly faster) then the switch-case? Specifically discuss control hazards and branch prediction (using branch-misses data)

|  |
| --- |
| In if case it is easier to correctly predict the path than in the switch. Also we can see that even if the instructions are higher, CPI is also higher in **if.** In the case of if, we can see less branches missed , therefore we can assume that **if** program run way smoother than **switch**. |

View the recorded perf trace using the command perf report -i perf\_switch.data. Copy-paste a screenshot of perf showing the runtime profile for the program using the switch version of the benchmark below

A screenshot of a computer

Description automatically generated

|  |
| --- |
|  |

View the recorded perf trace using the command perf report -i perf\_if.data. Copy-paste a screenshot of perf showing the runtime profile for the program using the if version of the benchmark below:

|  |
| --- |
|  |

# 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**.