Canvas Group Name: Student 1: Student 2:

CSE120 HA1: Performance Detective

Due Date: Sunday 1/24/23

This assignment will teach you to measure the performance of a real-world application. The assignment states what needs to be measured and provides some hints about the tools to be used, but it is your responsibility to determine the tools and methodologies to obtain the results. This assignment will require you to navigate a Unix operating system, compile a C program, and to profile its performance. You may utilize man, your favorite Unix textbook, or search engine to find out how to use these tools if you are not familiar.

Instructions:
To run the experiments, login via ssh to “bohr1.soe.ucsc.edu”. You need to be on the campus network or login to an accessible gateway first such as unix.ucsc.edu. Use the following user name to access the machine: cse120_${group} where ${group} is your canvas group number. (e.g. if you are group 42, use cse120_42 as user. Utilize the following password, “1xyzR4D8” which you NEED to change (`passwd`) after you have logged in the first time. As multiple students will be sharing the same machine, run your tests multiple times until you get reproducible results. If you need to copy a locally downloaded file to the server, use scp. You can also run the tests on your own/other Linux machine as long as it has 8 or more cores (you might need sudo access to run perf on your own machine).
Clone the video transcoder x264 via git (https://code.videolan.org/videolan/x264). If you are having trouble, export GIT_SSL_NO_VERIFY=1 may help.
Use ./configure and make to compile two different binaries of the same program. Compile one binary using no extra flags (*regular*) and another binary that disables platform-specific assembly optimizations (*disable-asm*). Read the configure script to determine how to pass such additional parameters to make. You may also need to pass two more flags when executing configure to compile successfully: "--enable-pic --enable-shared"
If you have any issues in doing this, checkout "man git", "man make", "man gcc"
Download the "paris_cif.y4m" video file from https://media.xiph.org/video/derf/y4m/

Submission:

Submit the answers to the questions below on Canvas as a single PDF per group.

1) Execution Time:

Run the two x264 binaries in single-threaded mode (--threads=1) to transcode the "paris_cif.y4m" video file.
Use `time` to measure the execution time of both binaries. What is the speedup you achieve by compiling without *disable-asm*? Run each test 5 times and compute the average.

Time *regular*:

Time *disable-asm*:

Speedup:

(4 Points)

2) Parallel Speedup:

Run the two x264 binaries with 1,2,4 and 8 threads. What speedups do you achieve? Plot a speedup graph showing threads on the x-axis and speedup on the y-axis for both binaries. Show 2 curves in the same plot, one for the *regular* compiled binary and another for the *disable-asm* binary.

(4 Points)

3) Amdahl's Law:

Provide the Amdahl's Law formula below. Compare the execution time of the 1 and 8 thread execution for the regular binary. What is the parallelizable fraction of the x264 application according to Amdahl's law?

(2 Points)

4. Perf Record

Using a single thread, use perf record to determine where the application spends most of its time. Determine the 4 function names that consume the most clock cycles for the *disable-asm* binary. Then observe the same functions with perf using the *regular* binary. How much of the total execution time is spent in these functions? Fill in the table below:q

Function	%*disable-asm*	%*regular*	Execution time *regular*	Execution time *disable-asm*

(4 Points)

5. Perf Stat

Using a single thread, use perf stat to determine the instruction count and average CPI for the two compiled binaries. Compute the clock cycle time of the processor.

Optimization	Exec time	IC	CPI	Cycle Time
regular				
disable-asm				

(2 Points)

6. Perf Topdown

Using a single thread, use perf stat's top-down methodology to determine the fraction of cycles spent on the frontend, backend, retired and bad speculation [1]. What is the best CPI that can be achieved (in theory) on this system? (The optimal case is where 100% cycles are in the retired category and 0% cycles are in the frontend, backend, mispeculation stall categories.)
HINT: Find a tool to pin the evaluated application to a specific core to enable interpretation of topdown results.

Frontend:

Backend:

Retired:

Bad Speculation:

Optimal CPI:

(5 Points)

[1] Ahmad Yasin: A Top-Down Method for Performance Analysis and Counters Architecture
https://drive.google.com/file/d/0B_SDNxjh2Wbcc0lWemFNSGMzLTA/view