COL380: 2022-23 Semester 2
Assignment 0

January 3, 2023

1 Introduction

In this assignment, you will learn how to profile code and its importance. You will use profiling tools to identify hotspots in the code, and iteratively optimise them till the desired efficiency is reached. You are provided with the base code for this assignment. The program reads numbers and ranges from **dfile** and **rfile** respectively. It then allots the numbers to their respective ranges. The program uses OpenMP for threading. You don't have to worry about that at the moment. You'll learn about OpenMP later on in the course. The aim of this exercise is to optimise the code for better efficiency. For identifying prospect optimisations, you'll use Perf.

Perf is a profiler tool for Linux 2.6+ based systems that abstracts away CPU hardware differences in Linux performance measurements and presents a simple commandline interface. Perf is based on the perf_events interface exported by recent versions of the Linux kernel. Here are some resources to get your feet wet: official tutorial, read-1, read-2, video-clip

Run the experiments as instructed, answer the questions in your writeup and attach screenshots wherever instructed.

Please run **chmod 700** ⟨*file*⟩ on files that you don't want other users on the cluster to see. This changes the read permissions of the specified file so that only you can read, write, and execute the specified file.

Note: In all the subsequent sections, you are only allowed to modify these files: **Makefile, classify.cpp and classify.h**.

2 Setting up and Running Perf

Perf is already installed on the css cluster that you have been given access to. In this section you will record perf data on the classify code and analyse it. Run **perf list** to see a list of all events that perf can record. To compile and run the program, use the makefile provided.

2.1 Perf stat

The perf stat command instruments and summarizes key CPU counters (PMCs). In this section, you will run perf stat on the given code and analyse the stat summary that perf outputs.

Vary the number of threads in the program and run perf stat. Try for values 1,4,8, 12,....,32 and plot the values of total time elapsed and cycles as a function of number of threads.

Do you observe any pattern in these plots? If yes, what is the prospective reason for it?

2.2 Perf Record

For subsequent sections, submit report files generated from runs made using 4 threads and 10 repetitions (unless explicitly mentioned otherwise).

1. Use **perf record** to record the perf data and name the generated report file **perf_1.data**. Without any event flags specified, cycles are recorded by default.
2. Use **perf report** to inspect the file you generated in the previous step. Use the annotate option in the report to obtain hotspots at the instruction level of a function. Alternatively, you can also use **perf annotate** to directly view annotated assembly code.
3. Which assembly instruction takes the most CPU time?
4. Can you map the instruction to the part of source code it corresponds to?
5. Tweak the makefile to allow the perf report to show the source code along with the assembly instructions. Run **make clean** before recompiling.

3 Hotspot Analysis

At this point, your perf reports should indicate which part of the source code runs for what percentage of time.

1. Run **perf record** and name the generated report file **perf_2_1.data**.
2. Report the top hotspot in your write-up (attach a screenshot of the perf report showing the code and the percentage of time taken).
3. What is the prospective problem which makes this code snippet the top hotspot?
4. Can the code be optimised to improve performance of this hotspot?
   (a) If it can be optimized, suggest the optimisations in the write-up.
   (b) If it can't be optimized, reason why in your write-up.
5. By default perf records data for cpu-cycles only. Now, we will try to record more events through perf. Run **perf record** again to include the following information in the generated data: **branch instructions, branch misses, cache misses, page faults, cpu cycles**. Submit the report file as **perf_2_2.data**.

4 Memory Profiling

In this part of the assignment, we would use Perf to identify areas of code which are cache-unfriendly. A program's run-time can be significantly increased in case of high cache miss rate. Often, code can be written for better cache usage. In class, you have studied false-sharing. The code provided to you has instances of **false-sharing**. Identify and optimize them.

Note: Do not make significant changes to the algorithm (do not make a significant change to the number of operations), for example, you should not go and make a hash table to count.

1. Run **perf mem record** on the given code, and analyse the report generated. Name the generated report file **perf_3.data**
2. Report the top 2 hotspots (attach a screenshot of the perf report showing the code and percentage of time taken).
3. Based on the hotspots you obtained, identify at least two issues in the code which makes it cache unfriendly.
4. Suggest improvements and implement them. Run **perf mem record** after each improvement and submit the screenshots of the hotspots identified earlier. Name the generated report **perf_4.data**

5. Run perf with the cache misses flag on the original code and the final code you obtain. Name the generated report files **perf_5_1.data perf_5_2.data** respectively. Do you see an improvement? If not, suggest ways to further improve cache hit rate.

5 Submission Instructions

1. Name your write-up: report.pdf.
2. Create a directory named ⟨*entry_no*⟩. The directory should contain: the final optimised source code (classify.cpp and classify.h); the perf reports; your write-up. Don't change the names of the files provided to you. For example, for some with entry number 2018CS50442, running the find command should have the following the output :

```
2018CS50442/
2018CS50442/A0
2018CS50442/A0/Makefile
2018CS50442/A0/classify.cpp
2018CS50442/A0/classify.h
2018CS50442/A0/perf_1.data
2018CS50442/A0/perf_2_1.data
2018CS50442/A0/perf_2_2.data
2018CS50442/A0/perf_3.data
2018CS50442/A0/perf_4.data
2018CS50442/A0/perf_5_1.data
2018CS50442/A0/perf_5_2.data
2018CS50442/report.pdf
```

3. Zip the folder and name the zip file: ⟨*entryno.zip*⟩.
4. Submit this zip file on Moodle.