CS 435/535 – Fall 2024

ÖZYEĞİN ÜNİVERSİTESİ

Homework 1

The deadline for this homework is **October 20th at 23:59 (midnight).** Please upload your solutions to LMS by the deadline.

(Step 0) Download and compile the code for sobel application given on LMS. For your convenience a Makefile is provided, so you can use it. If you haven't used makefiles before, you may check the following link for a quick tutorial: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

(Step 1) Install a proper profiler depending on your processor architecture (e.g., x86, ARM) to perform performance analysis for applications and systems. The following guideline is for x86-based systems where you may use Intel Vtune profiler (there might be other profilers that you may want to use, so you can pick your preferred profiler, however, it is expected that you perform all the analysis requested in the following questions).

Download and Install Intel Vtune on your x86-base Unix/Linux system. Please follow the instructions on:

https://www.intel.com/content/www/us/en/docs/vtune-profiler/installation-guide/2024-2/linux.html

and

https://www.intel.com/content/www/us/en/docs/vtune-profiler/get-started-guide/2024-2/linux-os.html

Question 1 (10 points)

Report the CPU topology of your machine. Explain the details of your topology (e.g., what does each feature indicates). You may use tools to obtain the topology of your CPU, such as "likwid-topology" and "lscpu". You may need to install likwid to use "likwid-topology", which I encourage as likwid has other tools that are useful for profiling as well (you may check https://github.com/RRZE-HPC/likwid for details).

Question 2 (20 points)

There are two versions of the sobel filter implementation provided at step 0. The first version is serial version (i.e., sobel_seq.c) in which there is no parallelism. The second version is the parallel version (i.e., sobel_par.c) which you will implement in the next question. After you compile the serial version of the sobel filter (i.e., sobel_seq.c), please perform the following analysis through the profiler that you installed in step 1 (e.g., Intel Vtune – if you are using Intel Vtune, then you can follow the guidelines provided by the new project wizard. You need to select the location of the application and proper application parameters to make the analysis properly. You can pick "input1.jpg" as an input which is also provided along with the sobel code).

a) Perform an analysis to see how much time the sequential version of sobel filter spends on various part of the code (e.g., in Intel Vtune that corresponds to "hotspots" analysis). Report the result of the analysis and explain which part of the program is the most time consuming.

b) Perform an analyis to see how much memory "consumption" the sequential version of sobel filter has, and report the findings.

c) Perform "memory access" analysis to identify cache misses and bandwidth usage of the sequential version of sobel filter, and report the findings.

Question 3 (30 points)

Based on the analysis you performed for the sequential version of sobel filter (i.e., sobel_seq.c) in Question 2, you should parallelize it by using pthreds (you need to decide which function to parallelize). You should use "sobel_par.c" file to implement the parallel version. The program should get the thread number from a command line as an argument (e.g., ./sobel_par input.jpg 4).Please submit your parallelized code, and make sure you added proper comments for each modification you have. Explain your parallelization strategy (e.g., one thread for each x operation).

Question 4 (40 points)

Perform the following analyis for the parallelized sobel code that you wrote in Question 3 (i.e. sobel_par.c).

a) Perform an analysis to see how much time the parallel version of sobel filter spends on various functions (e.g., "hotspots" analysis in Vtune) when different number of threads are used. You should perform separate analysis for thread counts of 2, 4, 8, 16 and 32. Report the results similar to above.

b) Perform "threading" analysis for thread counts of 2, 4, 8, 16 and 32, to see how efficiently an application uses available cores. Report the results similar to above.

c) Perform "memory access" analyis for thread counts of 2, 4, 8, 16 and 32. Report the results similar to above.