**CSE3033 Project3 Report**

The objective of this project was to implement a program using multithreading and synchronization techniques to compute the sum of square roots within a given range. This report outlines the implementation details, the approach taken, and the results obtained from the execution of the program.

We implemented the program in C language, utilizing the pthreads library for multithreading support. The program receives user inputs for the range of numbers and the number of threads to be used. Three methods were implemented to compute the sum of square roots:

Method 1: Concurrent updates to global\_sqrt\_sum without mutexes.

Method 2: Serial updates with a shared mutex to protect global\_sqrt\_sum.

Method 3: Local sum computation per thread, with a mutex protecting updates to global\_sqrt\_sum.

The code was structured into modular functions for thread creation, square root calculations, and synchronization. Extensive comments were added to clarify each step and improve readability.

**Execution Results(Also included in Excel file as well-organized table)**

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A computer screen with white text

Description automatically generated

**Project Questions**

1. **Which method(s) provide the correct result and why?**

Method 2 and 3 generates correct results because accessing global variable global\_sqrt\_sum by the more than one thread at the same time is not permitted. In addition, if the method 1 runs with 1 thread, it also generates correct result.

1. **Among the method(s) providing the correct result, which method is the fastest?**

Method 3 is the fastest method between all three results. Because threads can run almost independently by different cores.

**3. Among the method(s) providing the correct result, does increasing the number of threads always result in smaller total time? Discuss this considering the number of CPU cores available in your computer (in Linux, lscpu command provides the number of CPU cores available in your**

**computer).**

No. As the number of threads increases from 1 to 8, it observes a significant reduction in execution time, indicating efficient utilization of available cores to parallelize the workload. However, beyond 8 threads, the trend starts to plateau or even slightly increase. However, after 8 threads, execution times started to get longer or change slightly. The test computer has 8 core. The reason of this might be that with 8 physical cores, after reaching 8 threads, the operating system might start to multiplex threads on the available cores, leading to contention and increased overhead due to context switching.

**4. Are there any differences in user time/system time ratio of the processes as the number of threads increases? What could be the cause of these differences?**

According the our data, increasing the number of threads causes to increase user time / system time ratio because with a larger number of threads, the operating system might spend more time managing thread creation, scheduling, and context switching which increases the system time. This increased overhead can elevate system time relative to user time.

**Executing The Program**

A black background with white text

Description automatically generated