Numerical Integration and Multithreading – Exercise I **– Riemann sum**

High Performance Computing for Science and Engineering I

Evgeni Todorov – 15-909-211

The code ran is present in the .zip file. The configuration of the machine is as follows:

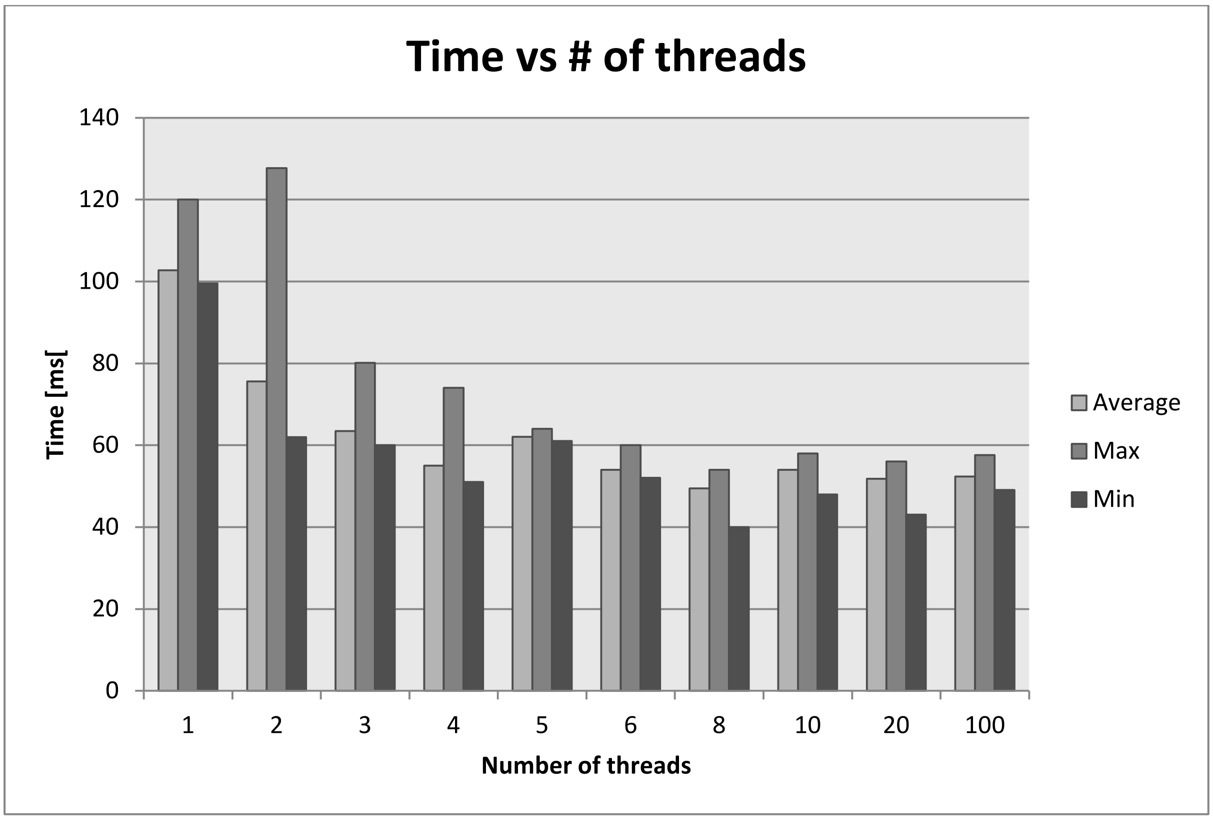
- OS: Windows 8.1
- CPU: Intel Core™ i5 3320M, Dual core, Clock: 2.6 GHz, boost 3.2 GHz, L3 cache 3 MB
- RAM: 8 GB

The compiler used was the MinGW32 version of g++. Additional parameters supplied during compilation: -std=c++11 (use C++11 standard); -O3 (full optimization for speed)

NB: Since MinGW32.g++ does not natively support threads, an open-source header file <mingw.thread.h> was used (https://github.com/meganz/mingw-std-threads).

As a performace/accuracy compromise, the **double** type was used for any calculations. Based on trial runs, N=**3 000 000** was chosen as the appropriate number of steps. At this level of accuracy, the result agrees with the analytical computation until the **12** decimal place. To establish statistical relevance of the results, the measurements were repeated 100 times.

Figure 1 Dependence of the average, maximum and minimum time taken for the computation against the number of threads. 3 000 000 steps. 100 repetritions of the measurement.



Since the Core i5 3320M CPU used in the test has only two separate cores, the improvement from program parallelization is quite limited, to about 40% decrease in execution time. What is interesting to observe, however, is the dependence on the performance speed on prime divisors, powers of 2 having better performance even when the number of threads in increased above the number of cores, suggesting some smart work on part of the compiler and the OS. However, after more than ~ 4 threads/core the overhead of multithreading is just too high.