Instruction sheet for lab class 4

Exercise 1

Download the program find_max.cpp from moodle. It contains a function findMaxElement that iterates through a vector and finds the largest element.
Parallelize the function using 4 threads. The work must be evenly distributed between the threads.
Avoid race conditions by avoiding parallel access to shared data! For this exercise, do not use locks, mutexes or atomics or any other synchronization measures.

Use the thread sanitizer or helgrind to verify that there are no race conditions in your code.

Exercise 2

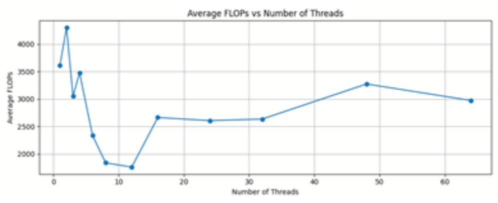
Again, consider the calculation of the vector triad.
Implement a parallel version of the computation, where each thread computes a portion of the vector. The initialization is carried out on thread 0 and the computation of the triad is done in parallel.

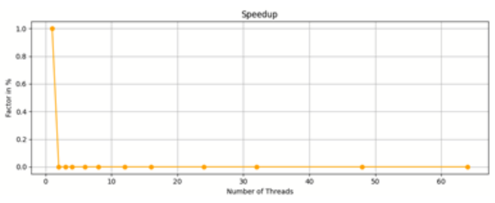
Benchmark your implementation for 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, and 64 parallel threads **on the cluster**. Use a size of $n = 2^{17} \cdot 9$, making the distribution of work to the threads easier.

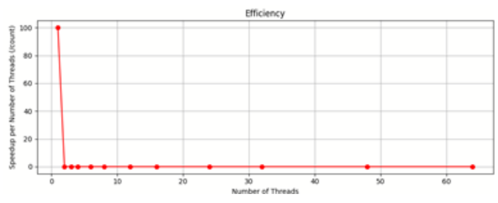
Then consider the following three quantities:

- Performance: floating point operations per second
- Speedup: runtime divided by the serial runtime
- Efficiency: speedup divided by number of threads

Generate plots of these three quantities as a function of thread count.







How many cores does the system you use have? How many cores share a socket? Does this help you to explain parts of your plot?

2 sockets, 6 cores share a socket, we can't really make sense of the graphics, for example at 12 threads the flops are really low and when using multiples of 6 we also don't observe any logical behavior.