**Report for counting ones:**

Summary: The running time of the program increased as the size of the array and the number of threads increased. For small sizes (1000,1000000), the running time was barely affected by the number of threads though it increased. However, for a very large input size like 1000000000, the running time significantly increased from 3.57 to 114.82 as the number of threads increased.

(The data in this graph is the mean of 30 runs of the program)

**Report of countRace**

Summary: for the count race, with the a small input like 1000, the program was almost computing the correct output. However, as the input size grows significantly, the program barely gives a correct output as shown in the plot below.

**Report of count\_mutex**

Summary: by allowing mutual exclusion using mutex, the program barely outputs the correct answer. On average, it’s giving the correct answer only ones. Unlike in count\_race, in which on average the program never outputs the correct result. As for the run time, although the input size and number of threads were increasing, the running time was decreasing which provides a good insight.

**Count\_private**

Summary: there is an improve in correctness, but not in performance. The program almost always gives the correct output on average. However, the running time was increasing with the input size and threads number. This is because of a probable occurrence of a deadlock as the number of threads increased, and it might be the case where there are unnecessary cache validations.

**Cache\_count**

**Summary:** there wasn’t any improvement in the performance of the program with this optimization. The program continues to almost always output the correct result.