Testing

Following a shared memory model using Posix threads:
Experimented with arrays of different sizes, namely 100, 10,000, 100,000, 1000,000, 10,000,000, 100,000,000, and 1,000,000,000 integers.
For each array size, experimented with different numbers of threads, namely 1, 2, 4, 8, 16, 32, and 64.

Specifications

Global variables are assumed.
Functions do not require anything, and return the number of 1s in their given array part/thread.

Data Analysis

Correctness

In the **race condition** program, the ratio of correct count over calculated count was acceptable for a small number of elements in the arrays (<100000). For larger sizes, the program returns obviously wrong counts at every run.
Why? Race condition happens when two processors(threads) access the same variable, and at least one writes to it. The accesses are not synchronized, so they could happen simultaneously. Thus, the data is inconsistent.
On the other hand, the other programs returned the right count at every run, no matter the size nor the number of threads, as we dealt with the race condition using a mutex, private variables, and padding caches:
The mutex, a special type of variable, is used to restrict access to a critical section to a single thread at a time. Thus, it guarantees that one thread "excludes" all other threads while it executes the critical section.
The private variables are not shared across threads, and syncing the results after the threads are done with their local workload. The program makes use of that through *private_count*, accessed privately by the corresponding threads.
Padding caches is used mainly to avoid extra work being done, by making the variables occupy all the core's cache line. In the program, this is done through a struct that includes an integer used as each thread's *private_count* and a dummy char array filling what remains of the cache line.

Execution time as a function of the Number of Threads & Array Size

Ranked by performance (increasing)

*We can clearly observe that the execution time greatly increases when the array size increases, no matter the strategy.
.Race condition: For arrays sized up to 100000, time is steadily increasing exponentially.
From there, with the increase of array size, time approaches to being a straight horizontal line (constant).
.Mutex: For arrays sized up to 100000, the time increases exponentially with the number of threads.
From there, the graphs start becoming interesting and approaching our goal, as starting from a thread number of 16, the execution time drops.
.Private counts: For arrays sized up to 1000000, time keeps increasing exponentially with the number of threads.
From there, we start observing a decrease in execution time with the increase of the number of threads. This decrease is way steeper than the one we saw using mutex, making its performance better than both the previous programs.