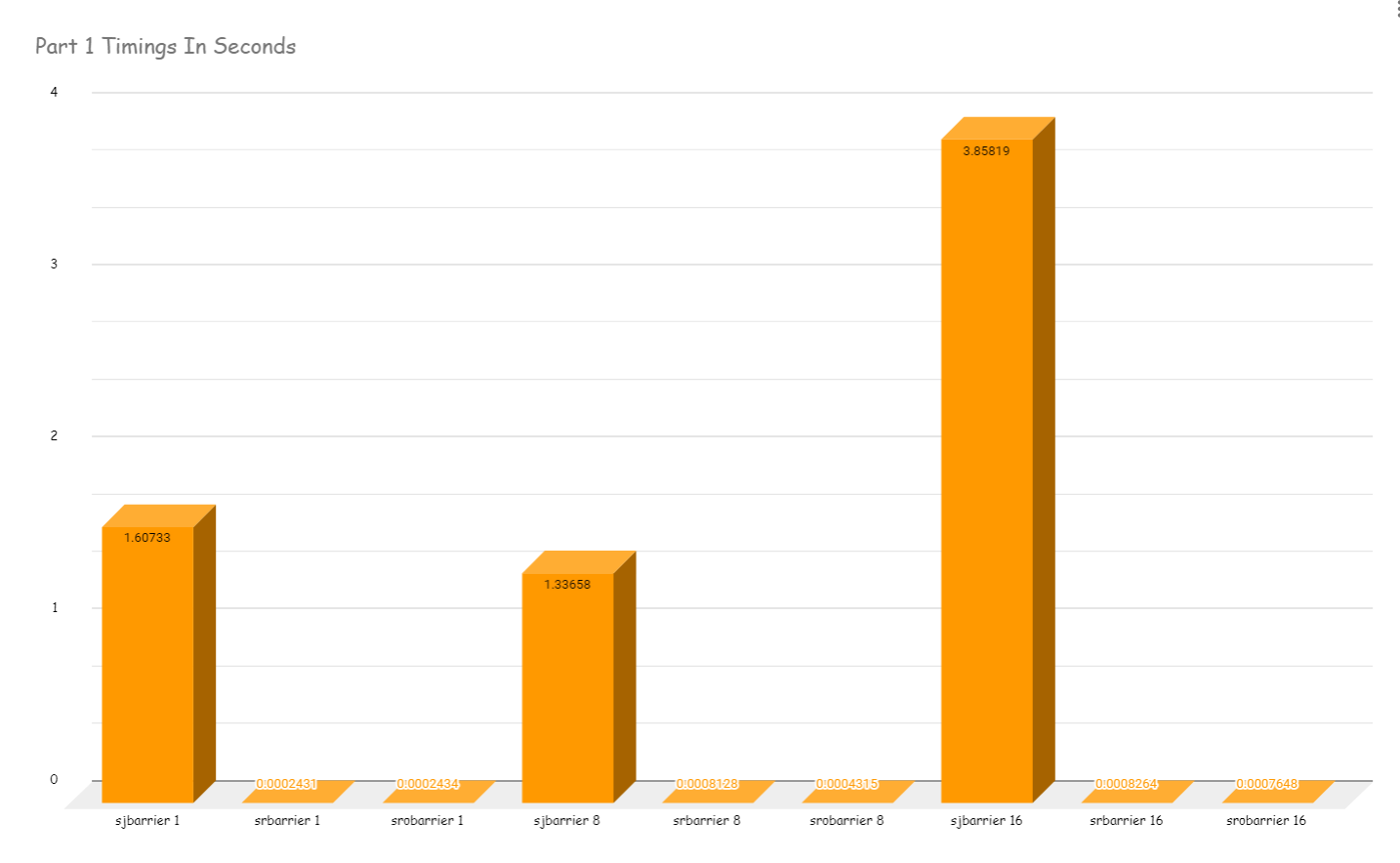
**Part1: Write Up**

For sjbarrier, my implementation consisted of chunking the array by the size of the array divided by the number of threads. For each iteration of my loop, I multiplied the chunk size by the iteration number and sent that to the repeated blur.

For srbarrier, my implementation of the barrier was almost identical to the book. Of course, not everything from java can be a direct translation into C++. That is where I decided to use atomic\_fetch\_sub()to calculate my position. I also used an atomic\_int and an atomic\_bool for my private variables.

For srobarrier, I was surprised in how simple our optimization was. I added this\_thread::yield()to my spin lock and memory\_order\_relaxed to my atomic load functions.



The results of these binaries were similar, even when the number of threads changed.

|  |  |  |
| --- | --- | --- |
| sjbarrier 1 | srbarrier 1 | srobarrier 1 |
| 1.60733 | 0.0002431 | 0.0002434 |

|  |  |  |
| --- | --- | --- |
| sjbarrier 8 | srbarrier 8 | srobarrier 8 |
| 1.33658 | 0.0008128 | 0.0004315 |

|  |  |  |
| --- | --- | --- |
| sjbarrier 16 | srbarrier 16 | srobarrier 16 |
| 3.85819 | 0.0008264 | 0.0007648 |

After I compared the time of each barrier implementation, it appears that the more threads we allocate, the more time it takes to synchronize.