Semaphore:

Relation between number of threads and performance:

|  |  |
| --- | --- |
| No of Threads | Time(in ms) |
| 1 | 304502 |
| 2 | 324185 |
| 3 | 1242263 |
| 4 | 1980344 |
| 5 | 2171719 |
| 6 | 2164567 |
| 7 | 2128849 |
| 8 | 1933908 |
| 9 | 1928915 |
| 10 | 1977185 |

From the figure, one can see that the performance of the counter scales poorly. Whereas a single thread can complete the million counter updates in a tiny amount of time, having two threads each update the counter one million times concurrently leads to a massive slowdown. It only gets worse with more threads.

Ideally, one would like to see the threads complete just as quickly on multiple processors as the single thread does on one. Achieving this end is called perfect scaling; even though more work is done, it is done in parallel, and hence the time taken to complete the task is not increased. Due to locks, the time increases.

Relation between number of threads and correctness:

|  |  |
| --- | --- |
| No of threads | Value |
| 1 | 1000000000 |
| 2 | 1000000000 |
| 3 | 1000000000 |
| 4 | 1000000000 |
| 5 | 1000000000 |
| 6 | 1000000000 |
| 7 | 1000000000 |
| 8 | 1000000000 |
| 9 | 1000000000 |
| 10 | 1000000000 |

From the figure, one can see that the counter is perfectly correct. It always reaches 1 billion if we use 1, 2 or 10 threads. Only one thread enters the critical region.

int sem\_wait( sem\_t \*s )

{

decrement the value of semaphore s by one wait if value of semaphore s is negative

}

int sem\_post( sem\_t \*s )

{

increment the value of semaphore s by one if there are one or more threads waiting, wake one

}