

MP2

Abhishek Johri



March 28, 2017

CS 484 Parallel Programming

University of Illinois at Urbana Champaign

* 1. Queue Implementation

Data (# of Threads are the variable)

|  |  |  |
| --- | --- | --- |
| Threads | Lock | Lockfree |
| 1 | 82395 | 103434 |
| 2 | 181426 | 139765 |
| 3 | 96813 | 179578 |
| 4 | 121699 | 212874 |
| 5 | 148130 | 254598 |
| 6 | 269514 | 293873 |
| 7 | 184096 | 326360 |
| 8 | 204204 | 360199 |
| 9 | 219269 | 448734 |
| 10 | 1269410 | 832801 |
| 20 | 459127 | 1182843 |
| 30 | 673968 | 1580033 |
| 40 | 932940 | 1950315 |
| 50 | 1060728 | 2048792 |

Graph

Data (# of insertions and deletions are variable):

|  |  |  |
| --- | --- | --- |
| Insertions/Deletions | Lock | Lockfree |
| 2 | 359383 | 655602 |
| 4 | 365914 | 695623 |
| 8 | 416644 | 695391 |
| 16 | 445435 | 786990 |
| 32 | 916307 | 849675 |
| 40 | 523851 | 888777 |
| 45 | 530709 | 924891 |

Graph:

Analysis:

In general we see that our lock implementation of the program is faster than out lockfree (atomic) implementation. This is because in the lock implementation all the threads are taking a common variable in this case a mutex lock. In the case of the lockfree version each thread has their own atomic variable that is created and maintained. Since they have their own variable it takes a longer time to add or delete whereas in the lock implementation the threads can do other things that do not require a lock and is a less serial process. I only used one atomic function which was atomic\_exchange(). I used this to create a lock for all the threads by making the entire critical section atomic to each thread. So in turns out that both of these programs are similar but it takes more time to create the atomic\_exchange than it is for creating a shared mutex.