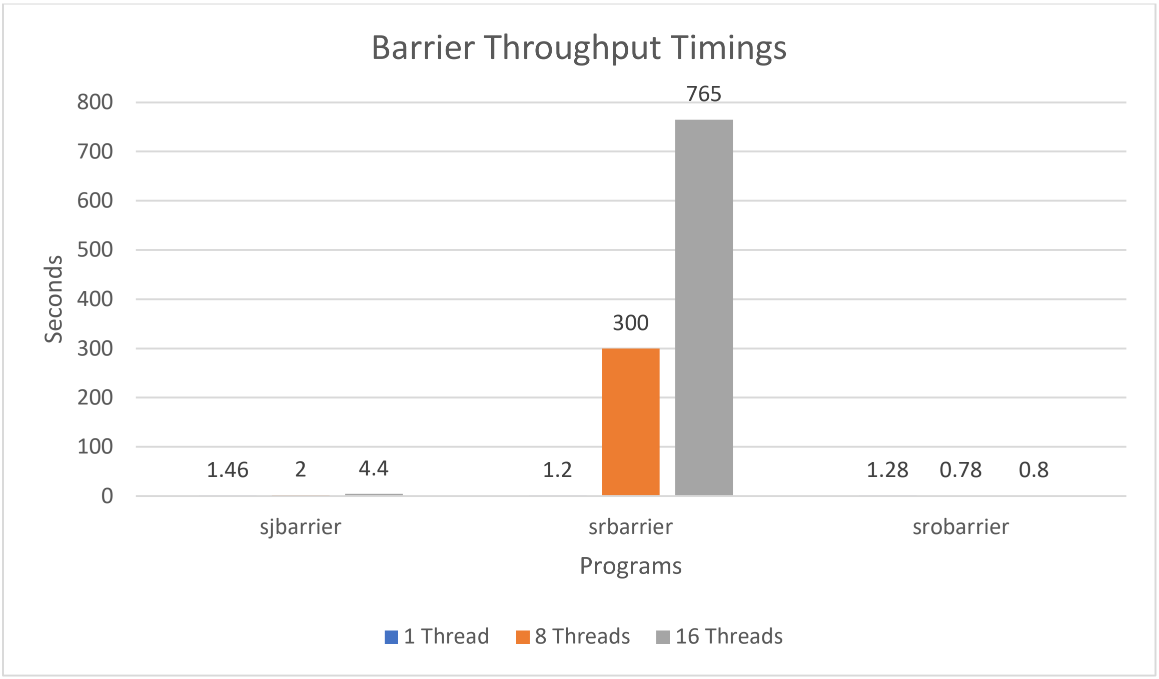
Homework 4 Write-Up

Galen Turoci

CSE 113

Part 1



Presented here are the timing results of 3 different barrier-based algorithms, though the first (sjbarrier) does not have a barrier at all. For reference, I am running this on an 8-core 3700MHz processor and 16GB of RAM. With the exception of srbarrier (8 and 16 threads), all of these results are based on the averages of running each implementation 5 times (srbarrier (8 and 16 threads) were run thrice each). The obvious outlier of srbarrier (8 and 16 threads) does concern me somewhat, but not overly so as I remember from both various piazza posts and lecture that barriers can increase runtimes, especially with greater numbers of threads. Before making the "sense" variable in my SRBarrier.h file atomic, these timing results were even worse, with the 8-threaded srbarrier taking around twice as long.

As we were given both the blur algorithm and the sense-reversing algorithm for this assignment, implementing both in our program was not too difficult, though it did take me some time to fully realize which variables in the barrier should be atomic and which shouldn't. After that, it was just a question of chunking out the blur algorithm between threads – easy enough since we'd done it before – and placing the barrier inside said blur; it is my belief that placing it at the end is correct, since at the end of the algorithm is when all of the algorithm's computations are finished.

Part 2

Part 2.1 Histogram:

```
histogram of different observations:
output0: 1 1: 262142
output1: 0 1: 1
output2: 1 0: 0
output3: 0 0: 1
relaxed behavior frequency: 3.8147e-06
```

Part 2.2 Histogram:

```
histogram of different observations:
output0: 1 1: 262142
output1: 0 1: 1
output2: 1 0: 0
output3: 0 0: 1
relaxed behavior frequency: 3.8147e-06
```

Part 2.3 Histogram:

```
histogram of different observations:
output0: 1 1: 262142
output1: 0 1: 1
output2: 1 0: 0
output3: 0 0: 1
relaxed behavior frequency: 3.8147e-06
```

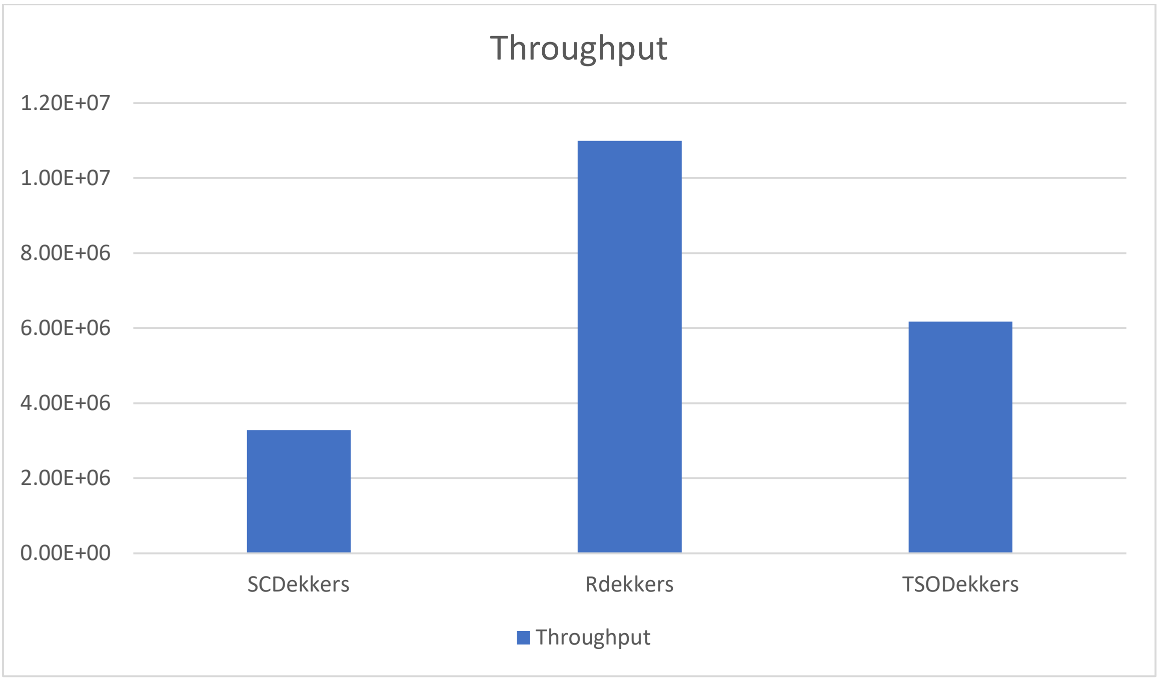
Presented here are the histogram results of 3 slightly different store-buffering litmus tests, each utilizing varying degrees of memory relaxation and barrier implementations. For reference, I am running this on an 8-core 3700MHz processor and 16GB of RAM. Each program was run 5 times. The results are not graphical due to the futility of doing so, as not only are the results the same for each part, but the "correct", or rather "expected" outcome was by far the most common. I am not certain why this is, as according to class materials, part 2.2 should have far more relaxed behaviors than parts 2.1 and 2.3. I can only conclude that my implementations were somehow wrong, though I am unable to see how.

Part 3

Part 3.1 Results:
throughput (critical sections per second): 3.2837e+06
number of critical sections: 6567391
number of mutual exclusion violations: 0
percent of times that mutual exclusion was violated: 0%

Part 3.2 Results:
throughput (critical sections per second): 1.09918e+07
number of critical sections: 21983516
number of mutual exclusion violations: 4163455
percent of times that mutual exclusion was violated: 18.939%

Part 3.3 Results:
throughput (critical sections per second): 6.17166e+06
number of critical sections: 12343322
number of mutual exclusion violations: 0
percent of times that mutual exclusion was violated: 0%



Presented here are the results of 3 different Dekker's mutex algorithm implementations, each utilizing varying degrees of memory relaxation. For reference, I am running this on an 8-core 3700MHz processor and 16GB of RAM. As expected, the relaxed memory implementation of the algorithm causes several mutual exclusion violations, at a rate of about 20%. Thankfully, however, with wisely-placed memory fences, these violations are corrected, and allow for almost twice as much throughput than the standard implementation.