CS 131 HW 3 Report

February 2025

1 Introduction

In this report, I will discuss the results of intentionally creating race conditions in the source code and its consequences. The purpose of this report is to observe the impact that relaxing the synchronization methods a program uses can have on its performance. To test this, I used a data structure with a swap method that increments one element of an array and decrements another.

2 Testing Environment

The experiment is running on Java ver. 23.0.2 on the SEASnet server, which has four Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz processors each of which have 4 cores.

To test the unsynchronized code I implemented the UnsynchronizedState and AcmeSafeState classes. UnsynchronziedState is identical to SynchronizedState except that the swap method does not have the synchronized keyword. AcmeSafeState on the other hand, implements synchronization using the Java ReentrantLock thread lock.

3 Discussion

The implementation for the SynchronizedState class is very reliable, with no mismatches observed when testing 10^6 and 10^8 swaps. However, its test durations were the highest, meaning it took the longest to perform the swaps. Its average swap time was worse than AcmeSafe when performing 10^6 swaps but for 10^8 swaps there were some cases where it performed slightly better. This could be because AcmeSafe needs to call functions for every operation which made the operations take longer.

AcmeSafe was also very reliable, no mismatches. Its testing durations were faster than Synchronized and for a majority of the test cases its average swap speed was faster, but for two cases it was not.

Unsynchronized was not reliable and there was data mismatched for every multithreaded case. However, it has the second fastest test durations and swap speeds for 10^6 swaps, and for 10^8 swaps it had the fastest in both categories. I believe that the increased speed with higher number of swaps is not worth using because the mismatch greatly increases with the number as well.

4 Measurements

	#Threads	#Swaps	#Arr Entries	Test Dur.	Avg. Swap Time
Synchronized	8	10^6	5	0.0699934s	559.947ns
			100	0.0830212s	664.169ns
	40	10^6	5	0.137191s	5487.66ns
			100	0.115136s	4605.44ns
Unsynchronized	8	10^6	5	0.158055s	1264.44ns
			100	0.146876s	1175.01ns
	40	10^6	5	0.0972058s	3888.23ns
			100	0.111050s	4441.98ns
AcmeSafe	8	10^6	5	0.0639082s	511.265ns
			100	0.0789365s	631.492ns
	40	10^6	5	0.0956727s	3826.91ns
			100	0.0905284s	3621.13ns
Synchronized	8	10^8	5	3.65649s	292.519ns
			100	3.41730s	273.384ns
	40	10^8	5	3.41730s	1747.00ns
			100	4.07273s	1629.09ns
Unsynchronized	8	10^8	5	2.56098s	204.878ns
			100	3.41730s	273.384ns
	40	10^8	5	2.49127s	996.506ns
			100	3.60357s	1441.43ns
AcmeSafe	8	10^8	5	3.74426s	299.541ns
			100	3.77577s	302.061ns
	40	10^8	5	3.70021s	1480.08ns
			100	3.73722s	1494.89ns