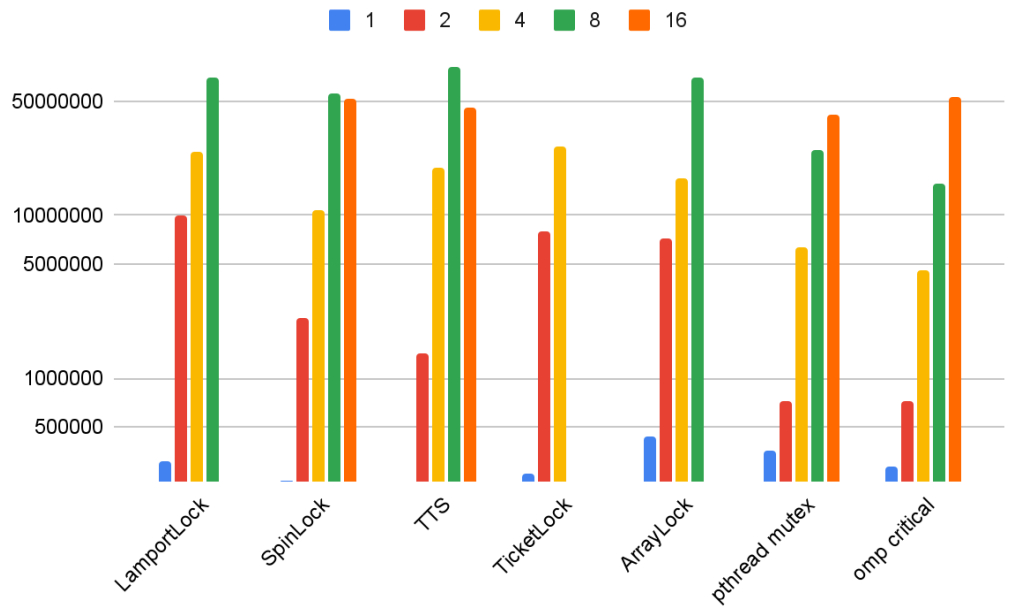
Assignment 2

Milan Anand Raj
200584

Implementing Locks



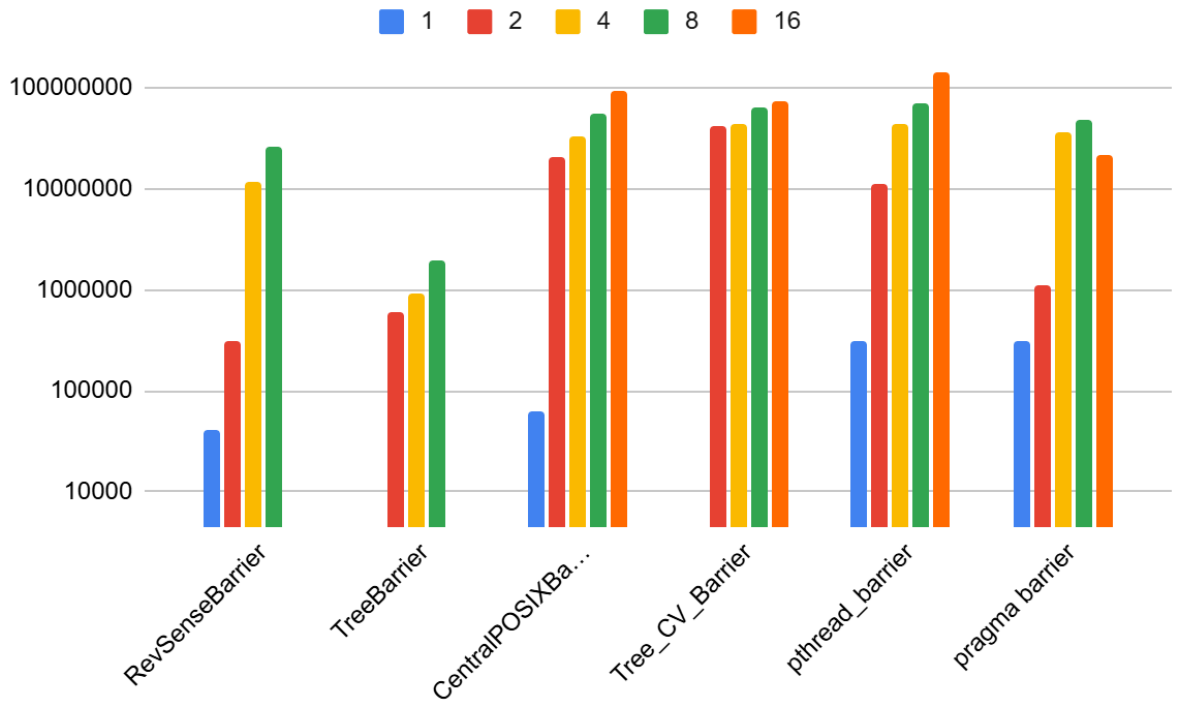
thread count	LamportLock	SpinLock	TTS	TicketLock	ArrayLock	pthread mutex	omp critical	Best performing
1	306129	235948	230271	256981	441288	360765	288038	TTS
2	10049558	2366231	1416766	8080306	7192279	721476	725081	pthread mutex
4	24372540	10791778	19793318	26300796	17071899	6398274	4646115	omp critritcal
8	70743320	56619929	81976207		69951051	25544894	15592123	omp critritcal
16		52002145				41395575	53729582	pthread mutex

Observations

- The performance of the Lamport Lock is poor, as expected, due to the high number of instructions involved, including the fence and memory clobber.
- In single-threaded scenarios, the Spinlock and TTS (Test-Test-Set) perform best with a thread count of 1. This is due to their lightweight nature, utilizing only the `cmpxchgl` assembly instruction to acquire the lock, which is expected to be fast without requiring cache coherence.
- For thread counts from 2 to 16, `omp critical` and `pthread mutex` exhibit the best performance among the locks tested.
- With two threads, TTS outperforms the Spinlock. TTS only attempts to acquire the lock when it detects it is free, thereby reducing cache coherence overhead compared to Spinlock.
- The Array Lock performs better than the Ticket Lock, largely due to fewer bus transactions.

Note: The lamport implementation is not working correctly in case of using ticket[tid] = *max_element(ticket, ticket+num_threads) + 1

Implementing barriers



thread count	RevSenseBarrier	TreeBarrier	CentralPOSIXBarrier	Tree_CV_Barrier	pthread_barrier	pragma barrier	Best barrier
1	41543	4461	63588	4573	317538	304501	TreeBarrier
2	308622	588751	20202267	40911726	11360777	1129690	TreeBarrier
4	11538657	900073	32330513	43382530	44357279	35441108	TreeBarrier
8	26370750	1935407	55427283	62667658	69884142	46954400	TreeBarrier
16			92633375	71942316	142610740	21834472	pragma barrier

Observations

- The Tree Barrier performs significantly better than other barriers up to a thread count of 8 due to its decentralized synchronization, which reduces latency to logP. This marks a substantial improvement over cases where false sharing was not avoided.
- However, the Tree Barrier shows poor performance when implemented with Conditional Variables (CV), as the increased scheduling and descheduling introduce heavy overhead.
- The centralized sense-reversal barrier performs worse than the Tree Barrier, as it incurs P latency compared to the logP latency of the Tree Barrier.
- The POSIX barrier's latency increases even further with its Conditional Variable implementation, primarily due to the additional overhead from scheduling and descheduling.