Heterogeneous Parallelism Mini Project

Title: Implementation of High Performance, Lock Free and Concurrent Data Structures

Team 1
Madhav Jivrajani (PES1201800028)
M S Akshatha Laxmi (PES1201800130)
Sparsh Temani (PES1201800284)

Introduction/Background

Spin Locks: Reloops till the CAS operations returns True

- Equivalent of acquire lock: **while(!lock.CAS(0, 1));**
- Equivalent of lock release: **lock = 0**
- While the thread is in the critical section, lock is set to 1, once it exists, one of the waiting threads sets the lock to 1 again using the CAS operation and enters the critical section.
- CAS (Compare and Swap) : Executed as a single instruction on the CPU (atomic)
- Can have severe performance implications as only thread can enter the critical section at once.

Issues with Spin Lock based Concurrent Data Structures

Reloops on Progress and Non Progress

- Uses additional CPU cycles even when no progress is made
  - When a thread with an acquired lock gets preempted, another thread which gets scheduled will waste CPU cycles

- If the thread with an acquired lock dies, there is no progress made
  - If a thread holding a lock dies in the middle, other threads waiting for the lock cannot proceed

- Priority inversion
  - A process with a lower priority is holding a lock that is required by a process of a higher priority

How Lockless Programming is useful ?

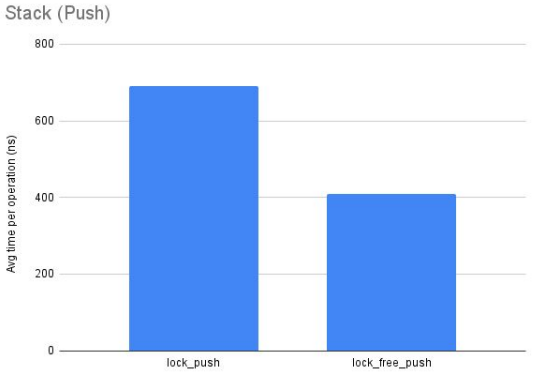
- Lockless programming makes sure at least one thread is making progress at any given instant (assuming it is scheduled by the OS).
- When the thread is preempted, another process can make progress since no lock is acquired
- Since no lock is held in any of the threads, if a thread dies, only the operations assigned to the thread do not execute where as the rest of the system can proceed as it is
- Lockless Programming guarantees system wide progress

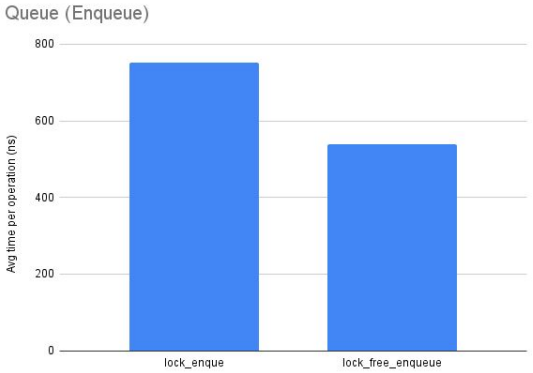
Implementation

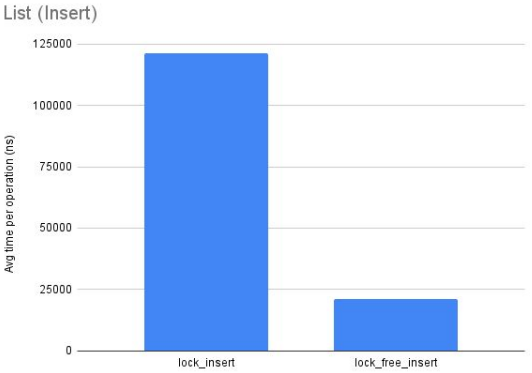
- Language used: Go
- Lock and equivalent lock-free implementations benchmarked using Go testing package, and checked for race conditions using Go's race detector.
- Call Graphs generated using `pprof` for the lock-free and its respective lock-based counterparts
- Data Structures implemented for mid-term review:
  - **Stack**: Push, Pop, Peek
  - **Queue**: Enqueue, Dequeue
  - **List**: Insert, Delete

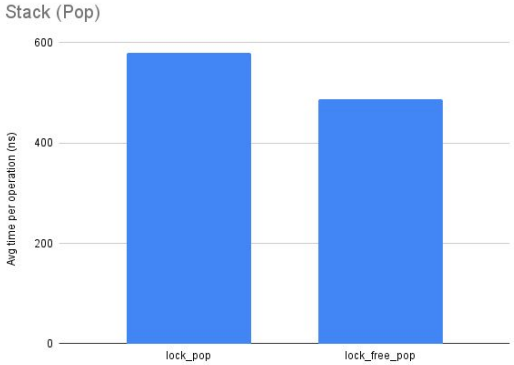
Quick Recap of Mid-term Review

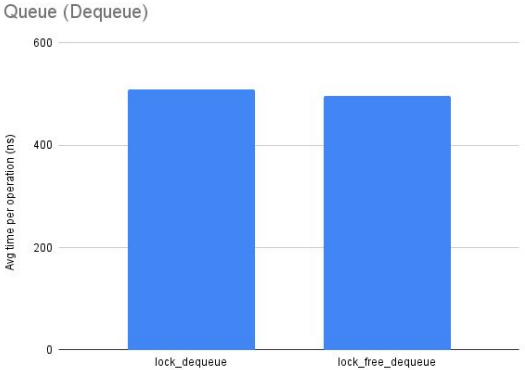
Averaged over 100,000 runs

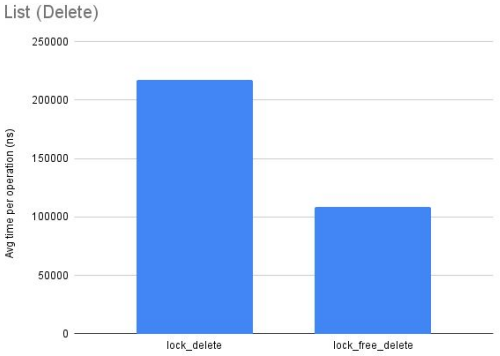










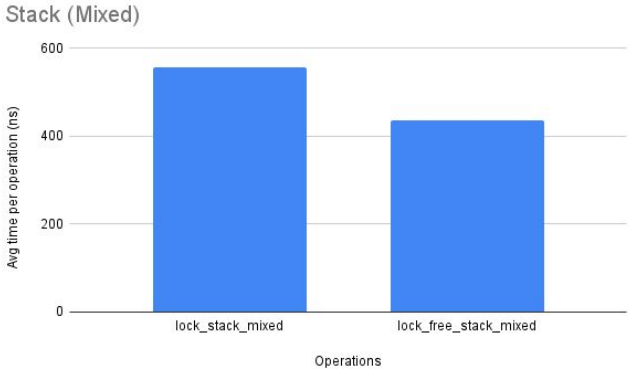


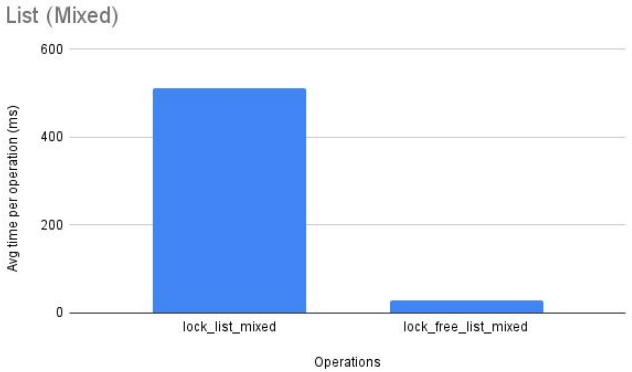
Progress Since Mid-term Review

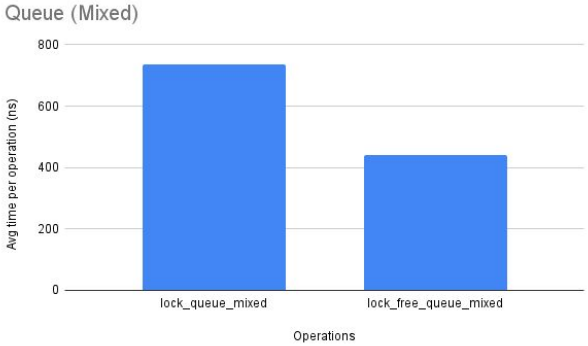
- Improve implementation of delete operation in list
- Profile to see if false sharing is happening
  - Try and reduce false sharing via padding memory
  - Analyse its impact on performance
- Further implementation:
  - Lock free map
- Benchmark for read + write happening simultaneously

Benchmarks for concurrent Reads and Writes

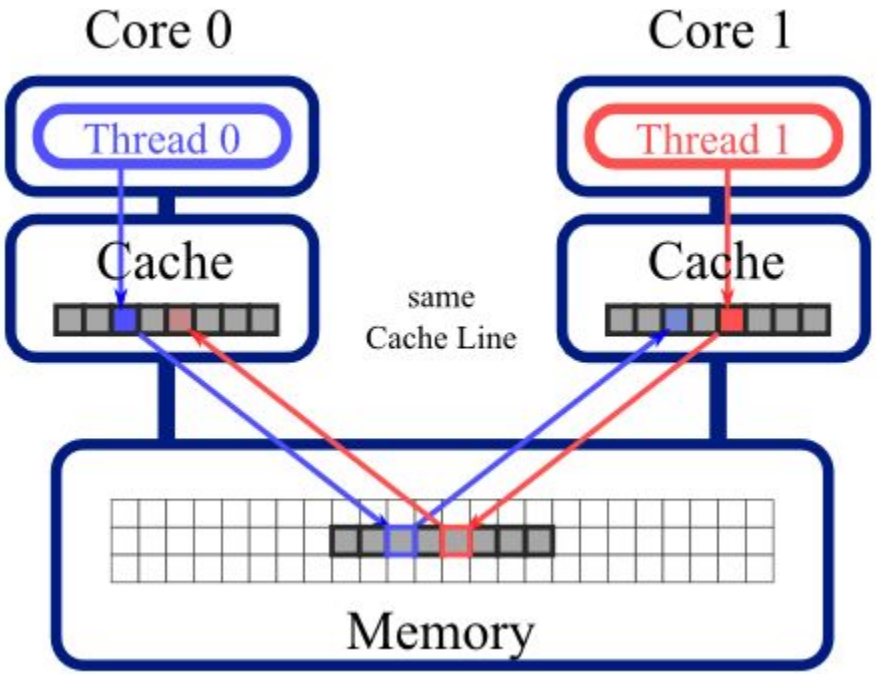
Led to some (really) interesting realisations (end of ppt).





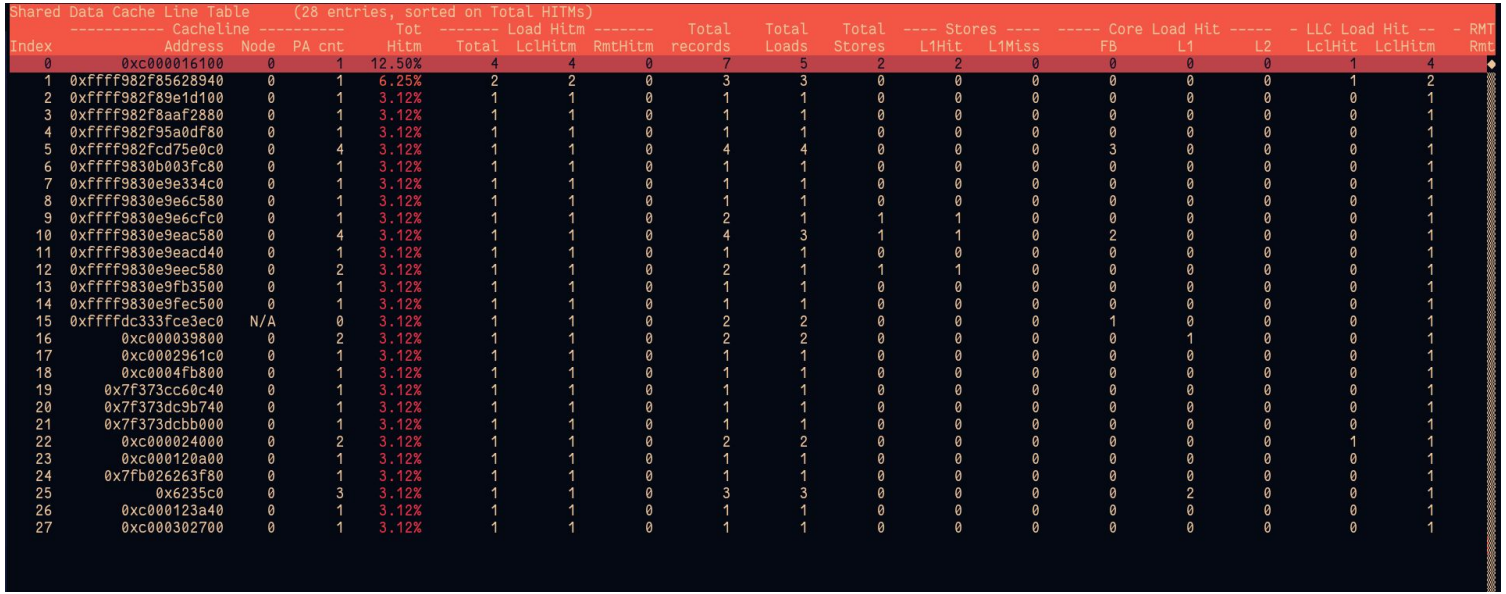


Cachelines and False Sharing



Profile for false sharing (and then cache miss)

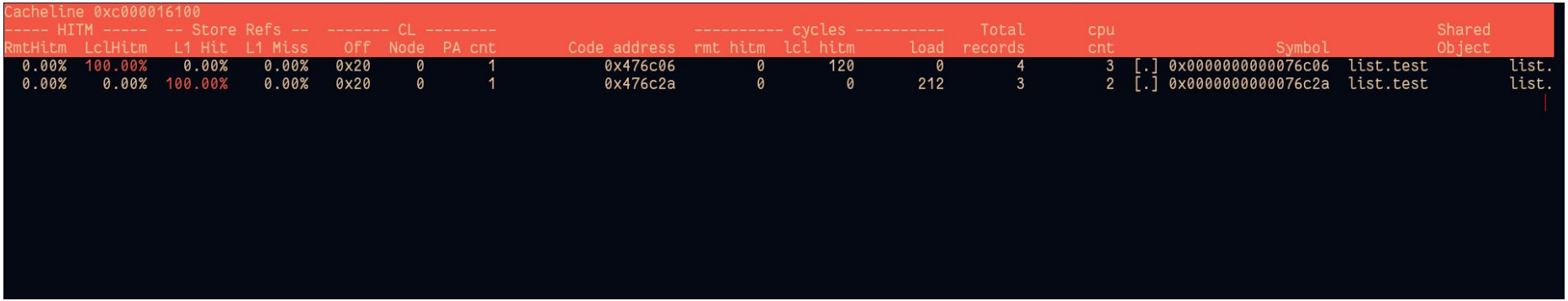
- Use `perf c2c` to detect false sharing.



Shared Data Cache Line Table (28 entries, sorted on Total HITMs)

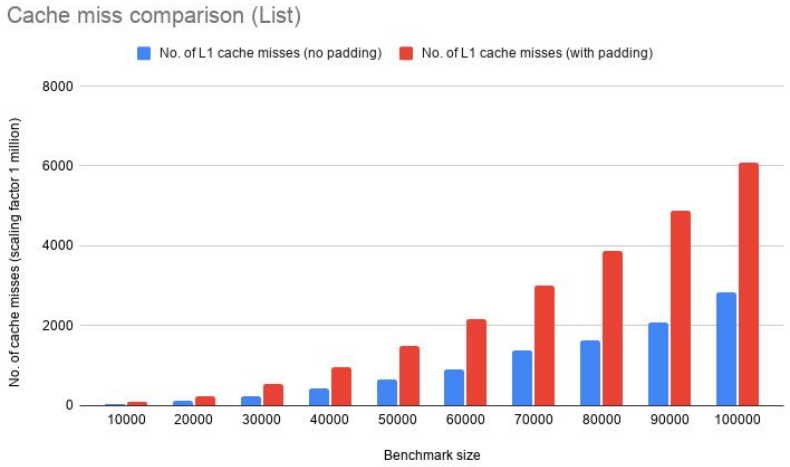
Index	Cacheline Address	Node	PA cnt	Tot Hitm	Load Hitm Total	Load Hitm LclHitm	Load Hitm RmtHitm	Total records	Total Loads	Total Stores	Stores L1Hit	Stores L1Miss	Core Load Hit FB	Core Load Hit L1	Core Load Hit L2	LLC Load Hit LclHit	LLC Load Hit LclHitm	RMT Rmt
0	0xc000016100	0	1	12.50%	4	4	0	7	5	2	2	0	0	0	0	1	4	
1	0xffff982f85628940	0	1	6.25%	2	2	0	3	3	0	0	0	0	0	0	1	2	
2	0xffff982f89e1d100	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
3	0xffff982f8aaf2880	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
4	0xffff982f95a0df80	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
5	0xffff982fcd75e0c0	0	4	3.12%	1	1	0	4	4	0	0	0	3	0	0	0	1	
6	0xffff9830b003fc80	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
7	0xffff9830e9e334c0	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
8	0xffff9830e9e6c580	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
9	0xffff9830e9e6cfc0	0	1	3.12%	1	1	0	2	1	1	1	0	0	0	0	0	1	
10	0xffff9830e9eac580	0	4	3.12%	1	1	0	4	3	1	1	0	2	0	0	0	1	
11	0xffff9830e9eacd40	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
12	0xffff9830e9eec580	0	2	3.12%	1	1	0	2	1	1	1	0	0	0	0	0	1	
13	0xffff9830e9fb3500	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
14	0xffff9830e9fec500	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
15	0xffffdc333fce3ec0	N/A	0	3.12%	1	1	0	2	2	0	0	0	1	0	0	0	1	
16	0xc000039800	0	2	3.12%	1	1	0	2	2	0	0	0	0	1	0	0	1	
17	0xc0002961c0	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
18	0xc0004fb800	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
19	0x7f373cc60c40	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
20	0x7f373dc9b740	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
21	0x7f373dcbb000	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
22	0xc000024000	0	2	3.12%	1	1	0	2	2	0	0	0	0	0	0	1	1	
23	0xc000120a00	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
24	0x7fb026263f80	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
25	0x6235c0	0	3	3.12%	1	1	0	3	3	0	0	0	0	2	0	0	1	
26	0xc000123a40	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	
27	0xc000302700	0	1	3.12%	1	1	0	1	1	0	0	0	0	0	0	0	1	

Profile for false sharing (and then cache miss)



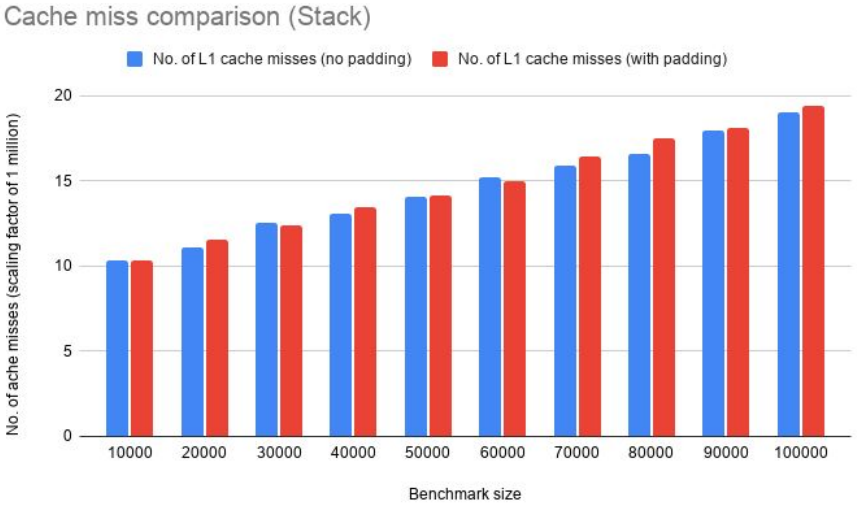
Profile for false sharing (and then cache miss)

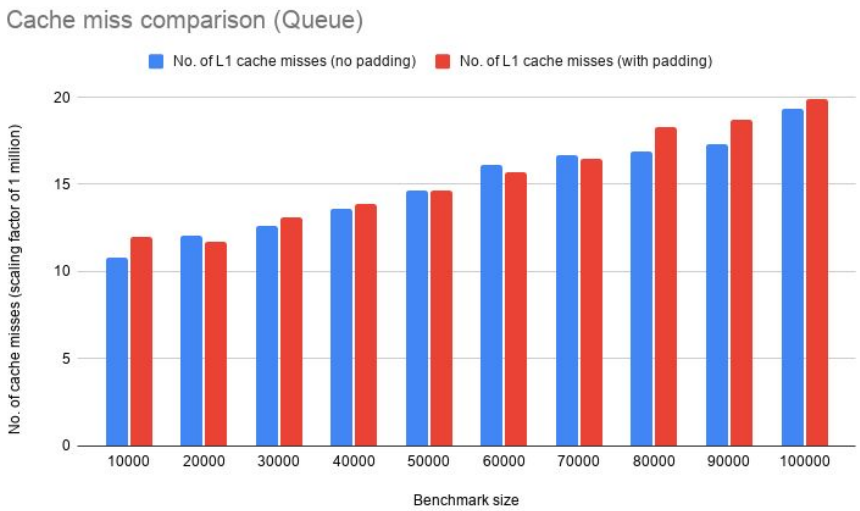
- Use padding to try and avoid false sharing.
  - Side effects of doing so in the case of operations which are *read heavy.*
  - Validating results by profiling for cache misses using `perf stat`
  - `cat /sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size`



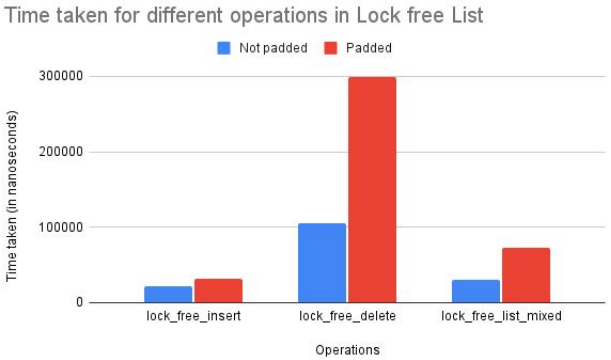

```
maddy@winston  index0
└─ $ ▶ cat /sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size
64
```

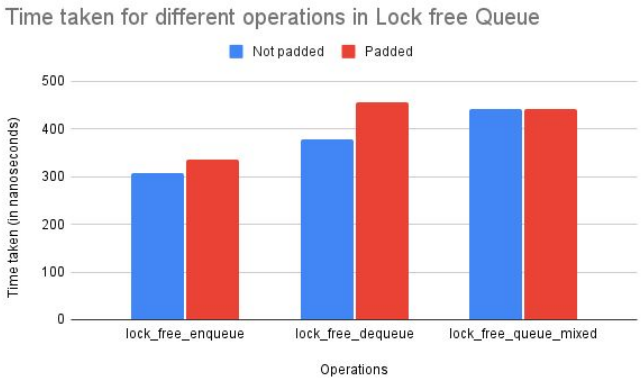
Profiling for cache miss

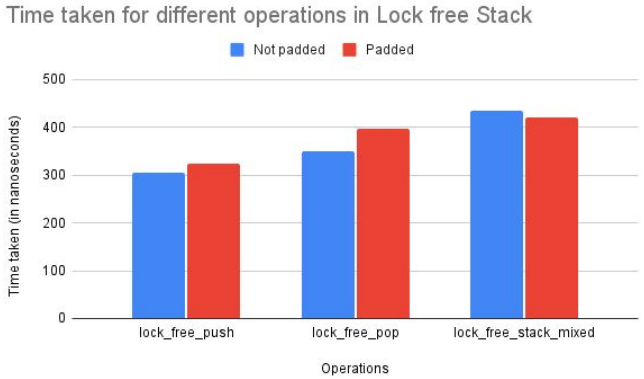




Operations with and without padding



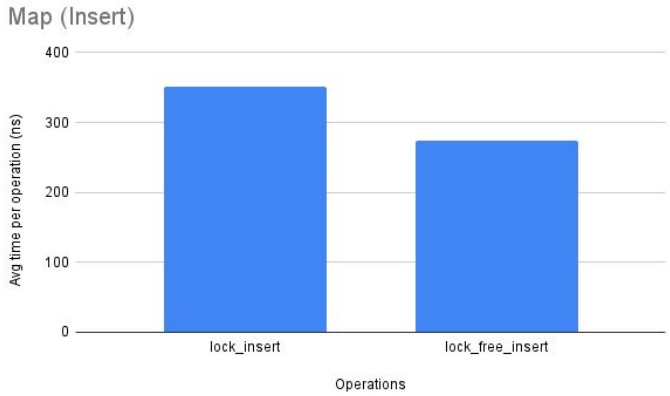


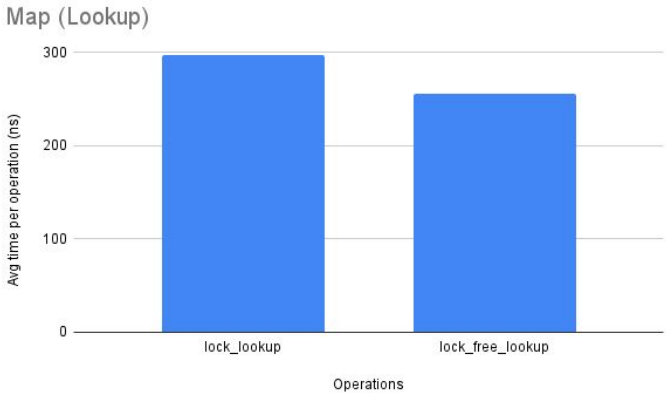


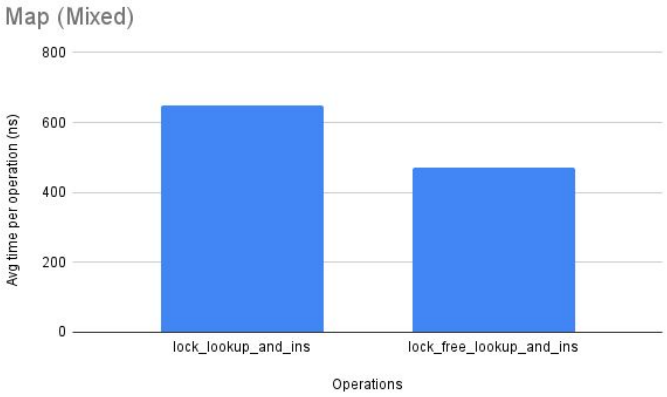
Lock Free Map (Methods)

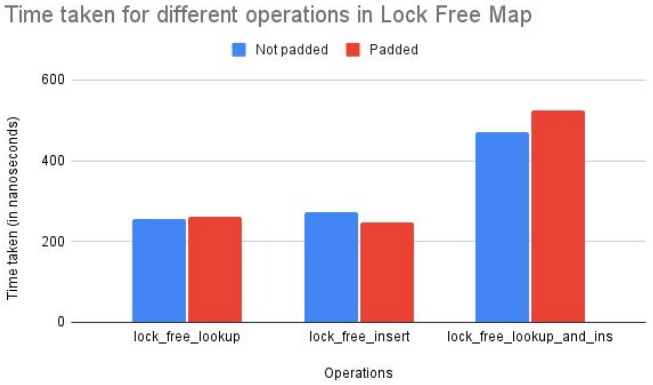
- `Insert`: Insert the key specified in the map
- `InsertIfDoesntExist`: Inserts if the existing value is ‘nil’
- `InsertCompare`: Takes a user defined function to decide if value should be inserted or not if the value already exists for the given key
- `Lookup`: If the element corresponding to a key exists, returns the element
- `Exists`: Checks is a key exists in the map or not

Lock free map

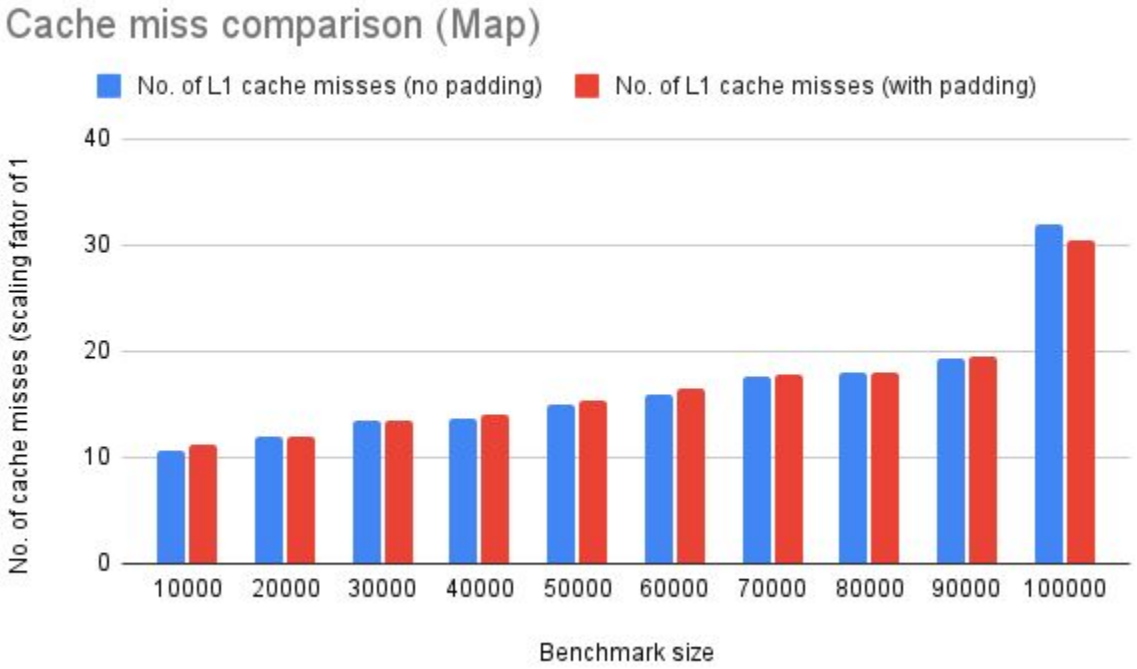








Lock free map



Miscellaneous Learnings

- Go race detector
  - Use of vector clocks for race condition detection
- Escape analysis
  - Can be seen using `-gcflags="-m"`
    - `"-m -m"` for more verbosity and so on.

```
./ops_test.go:74:19: int64(i) escapes to heap
./ops_test.go:82:19: int64(i) escapes to heap
./ops_test.go:95:19: BenchmarkLockDelAndIns ignoring self-assignment in queue.Tail = queue.Head
./ops_test.go:93:29: b does not escape
```

Work breakdown

Team member	Worked on	Time spent (approx.)
Sparh Temani	Stack, Map	15 hours
Madhav Jivrajani	Queue, False sharing	15 hours
M S Akshatha Laxmi	List, profiling	15 hours

- Note: the above breakdown mostly signifies obtaining metrics/raw data and implementation of some form; deriving insights from these raw metrics and reasoning about performance based on implementation, was collectively done by the team.

References

- **Implementing Lock Free Queues**, *J. D. Valois, Dept. of CSE, Rensselaer Polytechnic Institute.*
- **A Pragmatic Implementation of Non-Blocking Linked-Lists**, *Timothy L. Harris, University of Cambridge.*
- ***Introduction to Lock-free Programming*** *- Tony van Eerd, NDC Techtown*
- *Lock Free Programming - Herb Slutter, CppCon 2014*
- ***Designing a Lock-Free, Wait-Free Hash Map,*** *Shlomi Steinberg*
- ***A lock-free thread-safe HashMap optimized for fastest read access,*** *Cornel K*
- ***An intro to the Go race detector:*** https://www.youtube.com/watch?v=4r9Kr_HtGdI
- **`perf c2c`:** https://joemario.github.io/blog/2016/09/01/c2c-blog/

Thank you