Report

ES15BTECH11002, CS16BTECH11006

Design:

Number of threads and number of iterations taken from the user through terminal.

usleep function for delaying mymalloc request after each iteration.

The design of the algorithms is explained below.

2 approaches taken:

Naive implementation:

To make the program reentrant and thread safe, locks are created for each bucket in the bucket list.

Locks are acquired in the beginning and released at the end of mymalloc and myfree function.

By this approach we are able to parallelize requests for different nodes in the bucket.

But can we make it better?

A better implementation:

I implemented 2 locks for making mymalloc thread safe and reentrant.

This is how it works:

When a mymalloc request comes to a particular bucket in the bucket list, We lock the first 2 slabs in the head (proper care has been taken for edge case scenarios), we search the slabs for empty objects, if there exists, we return the pointer and release the acquired locks.
If the slabs do not contain any free objects we shift the locks to the next node, while releasing the older lock acquired.

Note: The 2 lock property is invariant while we search the nodes associated within the bucket, until we get a free object or we have to create a new slab.

When a myfree request comes to a particular bucket, we lock the corresponding slab from which the request have come.

If after performing the free, we find the slab to be empty(i.e, all object are free), we acquire the previous slabs' lock (this is to prevent deletion of wrong elements), after modifying the linked list, we release both the locks.

Difficulties:

Segmentation Faults, system_error : resource busy, system_error: invalid arguments

These errors were encountered while testing the program.

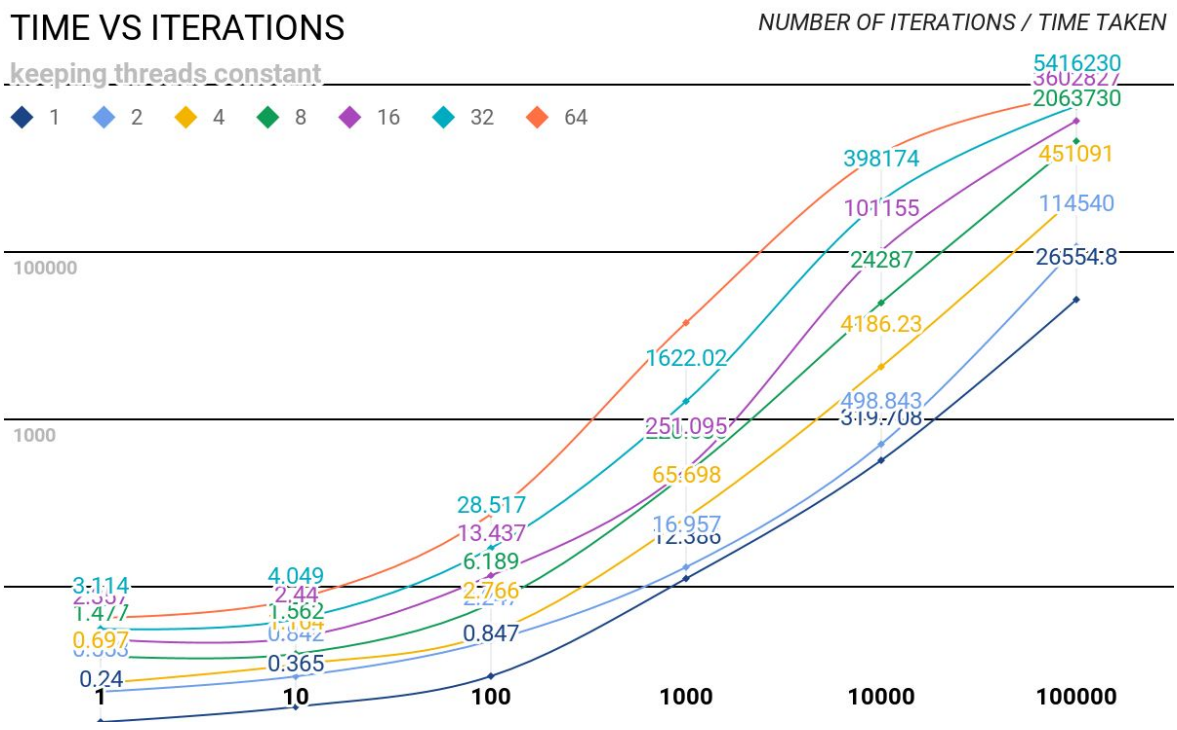
void* data was used at first which lead to segmentation fault, this issue was resolved by changing it to char data[0], the reason for this is not extremely clear to me, but a possible explanation is that c does not do range checking for arrays, and length greater than what is specified is allowed to be accessed without any compilation errors.

Care has been taken to unlock the mutexes in case of any failure to perform mmap.

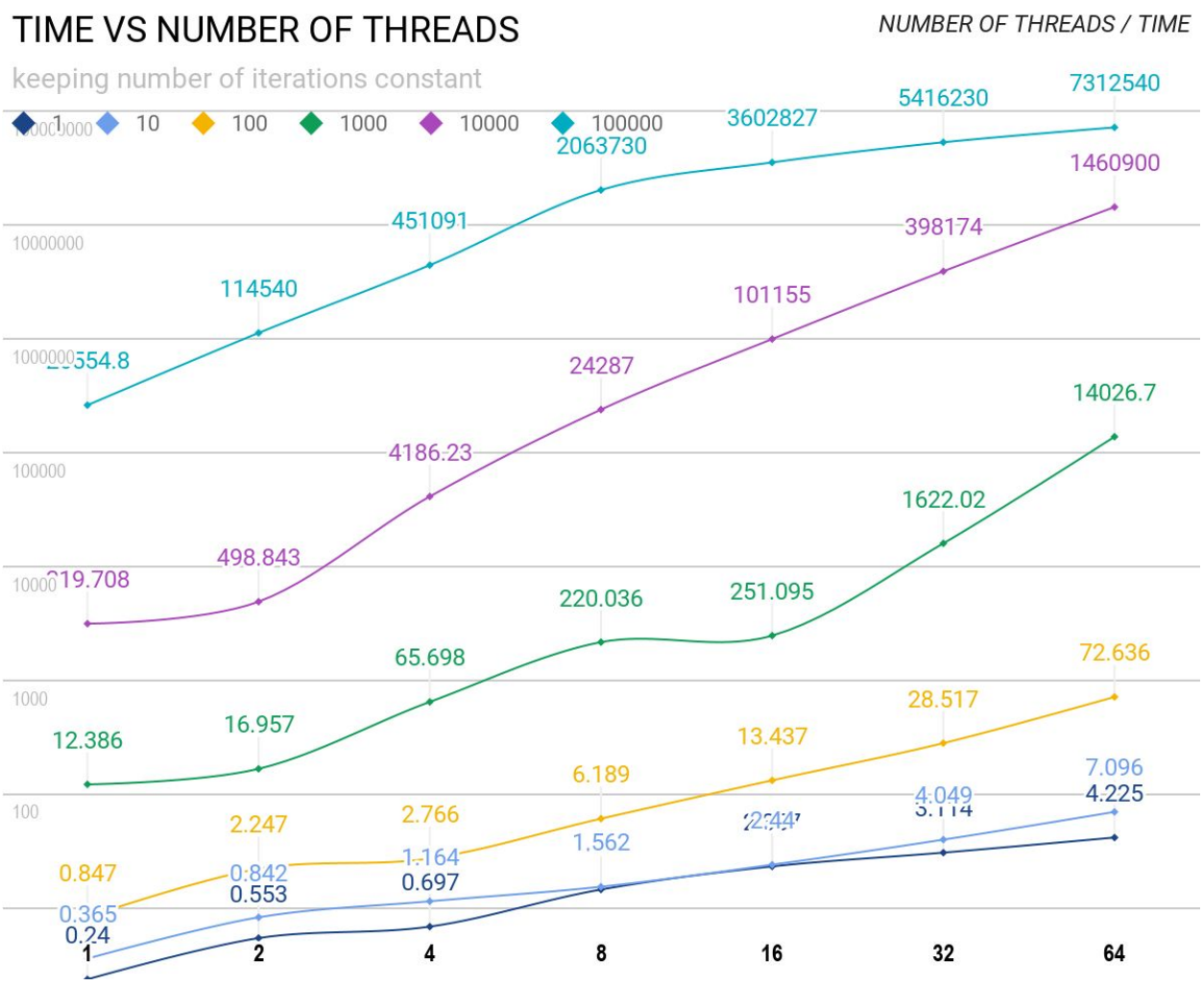
Graphs:

Naive implementation:

Time vs iterations

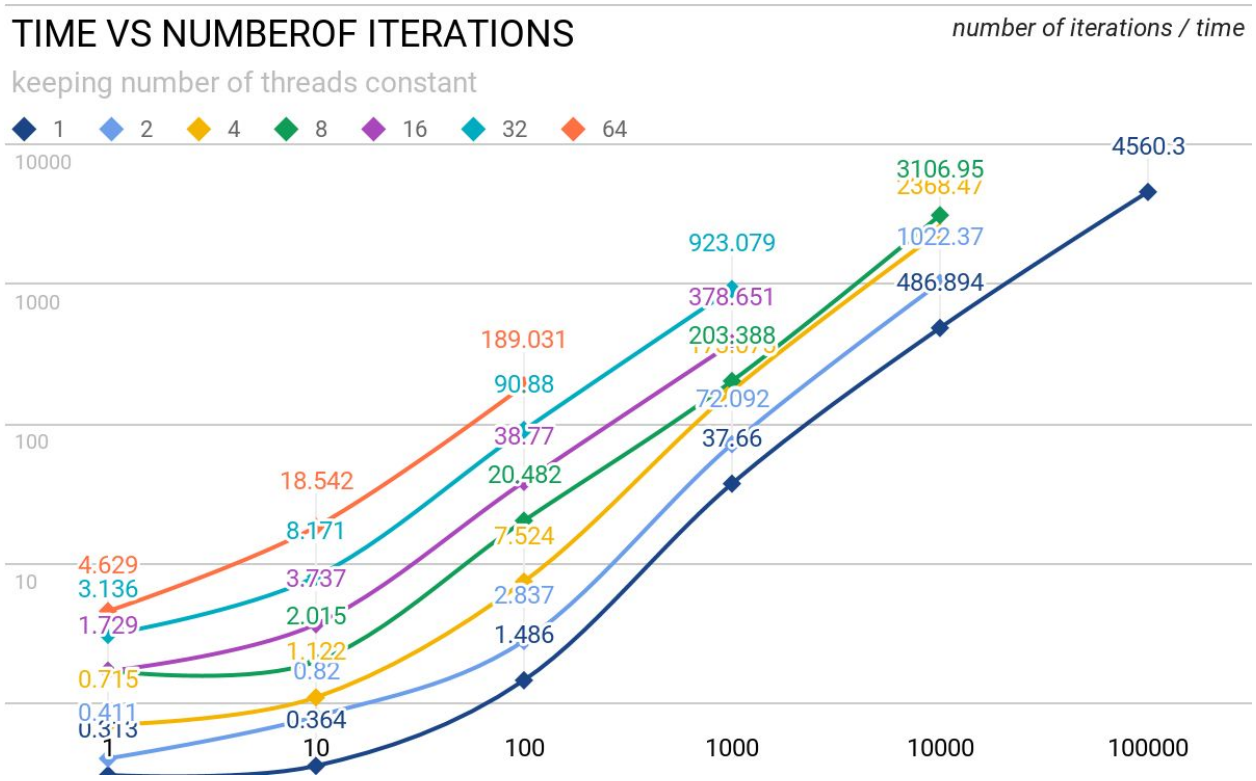


Time vs Threads

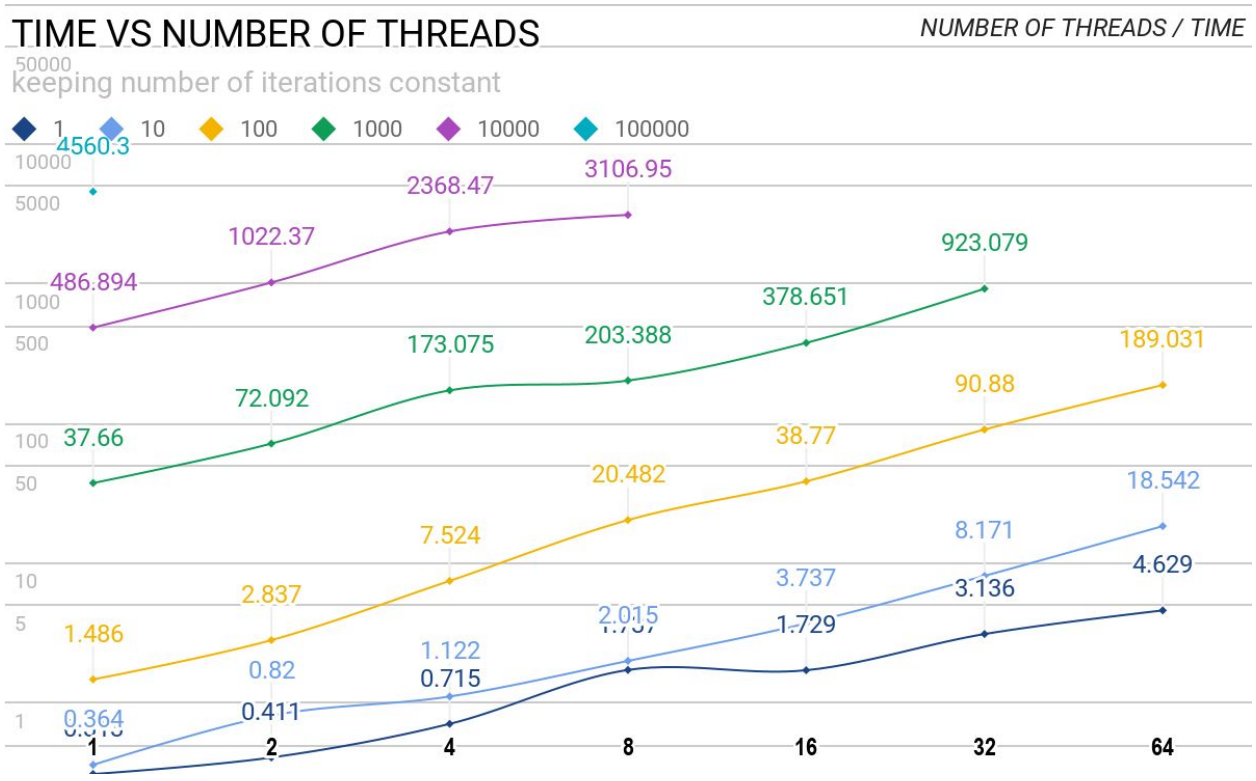


Efficient implementation:

Time vs iterations



Time vs Threads



Analysis:

No errors were encountered in the naive implementation of the program.

The better implementation was written in such a way so as to parallelize access to the same linked list, but as was observed from the performance, the naive works works better for the given scenario, maybe because 2 locks were needed to correctly perform mymalloc, and an additional lock (for a total of 2) were needed in case we needed to delete a slab, a seperate lock is needed anyways for each time myfree is called to lock the node getting processed.