# Problem – 2

# Open addressed hashtable with concurrent access

**Table of Contents**

**Abstract**

The problem involves implementing a concurrent open-addressed hashtable with quadratic probing that supports operations to initialize the hash table, add an element, delete an element, find an element and rehash the hash table. Rehashing is invoked internally if the load factor exceeds a given threshold L and it must be run in the background to ensure that other operations are locked out for the smallest time possible. The problem has been done using Pthreads(*POSIX Threads*).

**Usage Instructions**

The source code is in ‘hash_table_parallel.c’. To compile the code -
*gcc -w -fopenmp -pthread hash_table_parallel.c*

*[Make sure the ‘query’ file containing the input queries is present in the same directory before running. The total number of queries in the ‘query’ file provided in the test_cases_2 are* $10^5$ *but user can work with any number of queries by changing the NUM_QUERIES parameter in the code.]*

Then enter the value of ‘C’ and ‘L’ as prompted. Do ensure that value of L is between 0 and 1.

The result on the terminal displays the time taken by the code. The output file containing the result of queries executed by the threads will be created with name ‘*query_out_parallel*’. The actual queries (in the order executed by the threads) will be created in the file *‘query_actual’*.

The file used for testing is ‘*query*’ , which was generated randomly is available in folder ‘*test_cases_2*’. The format of the queries is :-

1. 0 key - find element with given key.
2. 1 key value - add element with given key and value.
3. 2 key - delete element with given key.

**Program Structure**

The program follows a multithreaded model using POSIX threads. It uses 4 threads as specified in the NUM_THREADS parameter in the program.

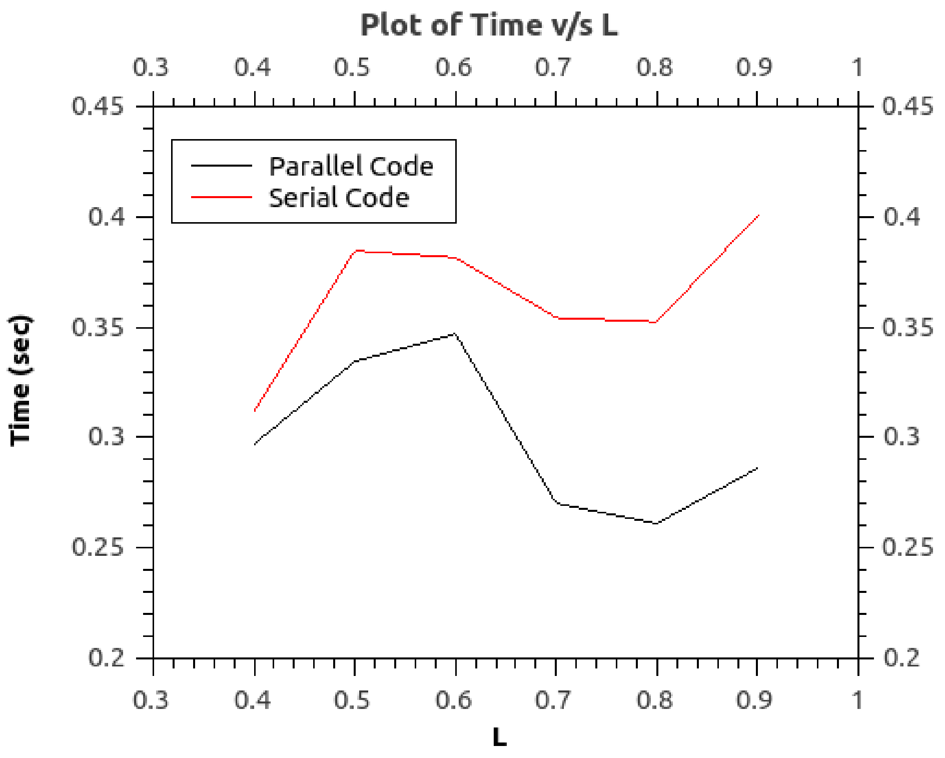
- Queries are randomly distributed to different threads taking care to ensure that no two threads get the same query.Each thread executes one of add (with rehash if necessary) , delete or find operation.
- During add,insert and delete, only those locations in the hash table that are being probed are locked instead of the entire hash table to increase concurrency. Once the location is probed , the location is unlocked.
- During rehashing, first all queries that are curently active are completed and then further queries are blocked only for the time that the hash table is being doubled in capacity. This keeps locking to a minimum.
- The output is printed in ‘*query_out_parallel*’ file. A separate lock is used for the printing process.

**Experimental Results**

Test data generation : A set of $10^5$ random queries containing roughly an equal number of find, add and delete operations was generated and saved in *query* file. The key and/or value paramters for each query are also generated randomly.

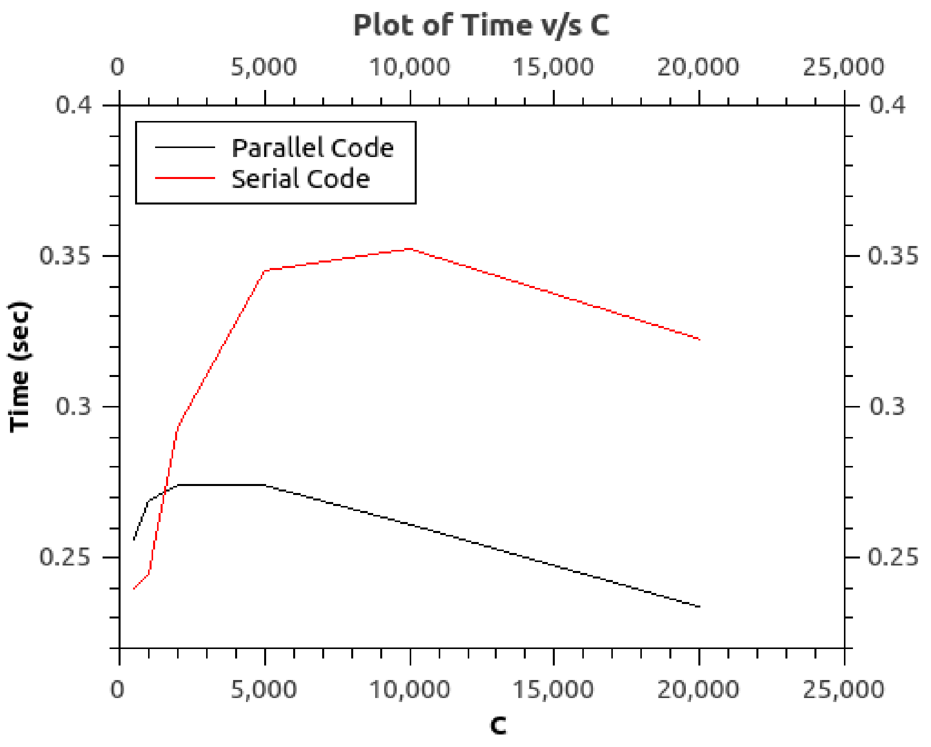
1. Variation of L keeping C constant

| **C** | **L** | **Serial code time (sec)** | **Parallel code time (sec)** | **Speedup** |
|---|---|---|---|---|
| 10000 | 0.9 | 0.400543 | 0.286105 | 1.40 |
| 10000 | 0.8 | 0.352421 | 0.261056 | 1.35 |
| 10000 | 0.7 | 0.354302 | 0.270232 | 1.31 |
| 10000 | 0.6 | 0.381605 | 0.346915 | 1.09 |
| 10000 | 0.5 | 0.384639 | 0.334467 | 1.15 |
| 10000 | 0.4 | 0.311860 | 0.297009 | 1.05 |



2. Variation of C keeping L constant

| C | L | Serial code time (sec) | Parallel code time (sec) | Speedup |
|---|---|---|---|---|
| 500 | 0.8 | 0.239671 | 0.256352 | 0.93 |
| 1000 | 0.8 | 0.244691 | 0.268918 | 0.91 |
| 2000 | 0.8 | 0.293326 | 0.274135 | 1.07 |
| 5000 | 0.8 | 0.345228 | 0.273990 | 1.26 |
| 10000 | 0.8 | 0.352421 | 0.261056 | 1.35 |
| 20000 | 0.8 | 0.322532 | 0.233720 | 1.38 |



**Best parallelism and speedup**

With C constant at 10000 , the best speedup obtained is 1.40 when L is 0.9.

With L constant at 0.8 , the best speedup obtained is 1.38 when C is 20000.

---