In-memory Data Structures Supporting Concurrency

Naif Jabir

What is concurrency?

- Concurrency is the execution of multiple instruction sequences at the same time.
- It happens in the operating system when there are several process threads running in parallel.
- The running process threads always communicate with each other through shared memory or message passing.

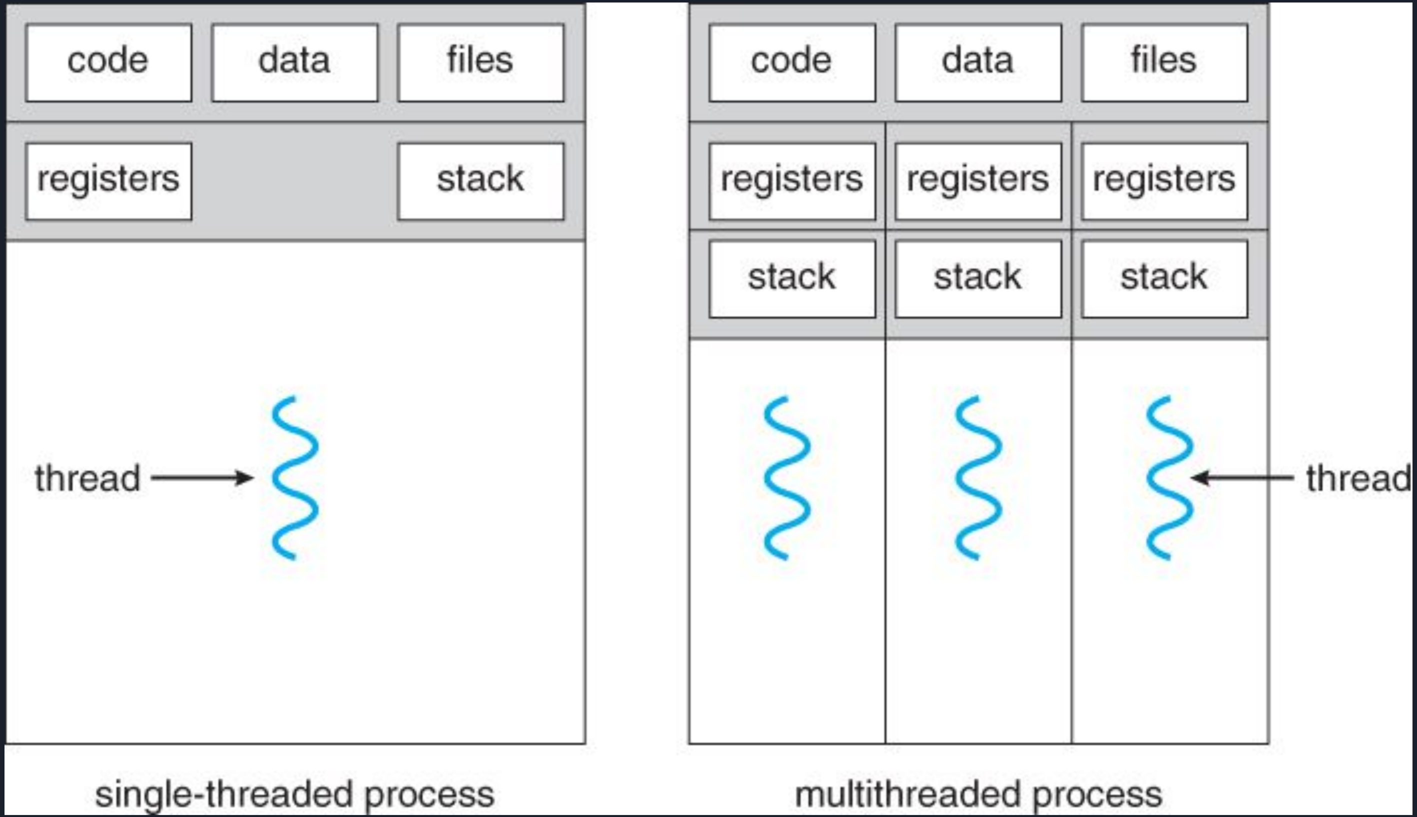
Models for concurrency

- There are two common models for concurrent programming: shared memory and message passing
- Shared memory model: modules interact by reading and writing shared objects in memory (same memory in same computer).
- Message passing: modules interact by sending messages to each other through a communication channel (same memory in different computers).

Types of concurrency

- Multitasking (execution of multiple tasks by rapidly switching between them)
- Multithreading (enable more than one user to run their processes)
- Multiprocessing (two or more central processing units within a single computer system)

Threads


code
data
files
registers
stack
thread
single-threaded process
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded process

Benefits of Concurrency

- Physical resource Sharing: Environment where we have multiple users since hardware resources are limited
- Logical resource Sharing: Shared file (same piece of information)
- Computation Speedup: Parallel execution
- Modularity: Divide system functions into separation processes

What is a data structure that can support concurrency?

- A concurrent data structure, is a data structure (e.g. list, stack) that can be used by multiple threads concurrently and it will always do the same type of process for each thread.
- Leads to more readable, maintainable, and reliable concurrent code

Advantages of Concurrency

- Running of multiple applications: It enable to run multiple applications at the same time.
- Better resource utilization: It enables that the resources that are unused by one application can be used for others.
- Better average response time: Without concurrency, each application has to be run to completion before next one.
- Better performance: Better Latency since processes have a chance to run without being blocked

Disadvantages of Concurrency

- Sharing global resources: Sharing of global resources safely is difficult. Two processes may target the same variables and order in which that happens is important.
- Optimal allocation of resources: It is difficult for the operating system to manage the allocation of resources optimally.
- Locking the channel: It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.
- Locating programming errors: It is very difficult to locate a programming error because reports are usually not reproducible (when i was experimenting i would run into a segfault, but after fixing it, i suddenly run into a memory allocation error)

Problems you will run into with concurrency

- Non-atomic: Operations that are non-atomic but interruptible by multiple processes can cause problems.
- Blocking: Processes can block waiting for resources.
- Starvation/Stagnation: It occurs when a process does not obtain service to progress.
- Deadlock: It occurs when two processes are blocked and hence neither can proceed to execute.

Code

```
std::unordered_map<std::string, int>
concurrentMap;
```

```
std::mutex mapMutex;
```

We have a hash table to help us access variables and do read and write

Mutex helps us protect our threads from accessing resources at the same time to prevent incorrect reads and writes (also avoid some segfaults and double freeing)

Code

```
void insertConcurrentMap(int start, int end) {
    for (int i = start; i < end; ++i) {
        std::string key = "Key" + std::to_string(i);
        int value = i;
        std::lock_guard<std::mutex> lock(mapMutex);
        concurrentMap[key] = value;
    }
}
```

Code

insertConcurrentMap: Initializes the concurrentMap by inserting key-value pairs within a specified range

Uses a std::lock_guard to lock the mutex during map modification, ensuring thread safety.

Code

```
void searchConcurrentMap (int start, int end, int
numSearches, std::vector< double>& latencies) {
    for (int search = 0; search < numSearches; ++search) {
        auto searchStartTime =
std::chrono::high_resolution_clock:: now();


        for (int i = start; i < end; ++i) {
            std::string key = "Key" + std::to_string(i);


            std::lock_guard<std::mutex> lock(mapMutex);
            auto it = concurrentMap. find(key);
            if (it != concurrentMap. end()) {}
```

Code

searchConcurrentMap: Performs multiple search operations within a specified range and measures the latency of each search.

We then record the latency in a vector

Code

Main:
initializes the concurrent map with key-value pairs using insertConcurrentMap

measures the performance of searching for these keys with varying numbers of threads (1, 2, 4, 8).

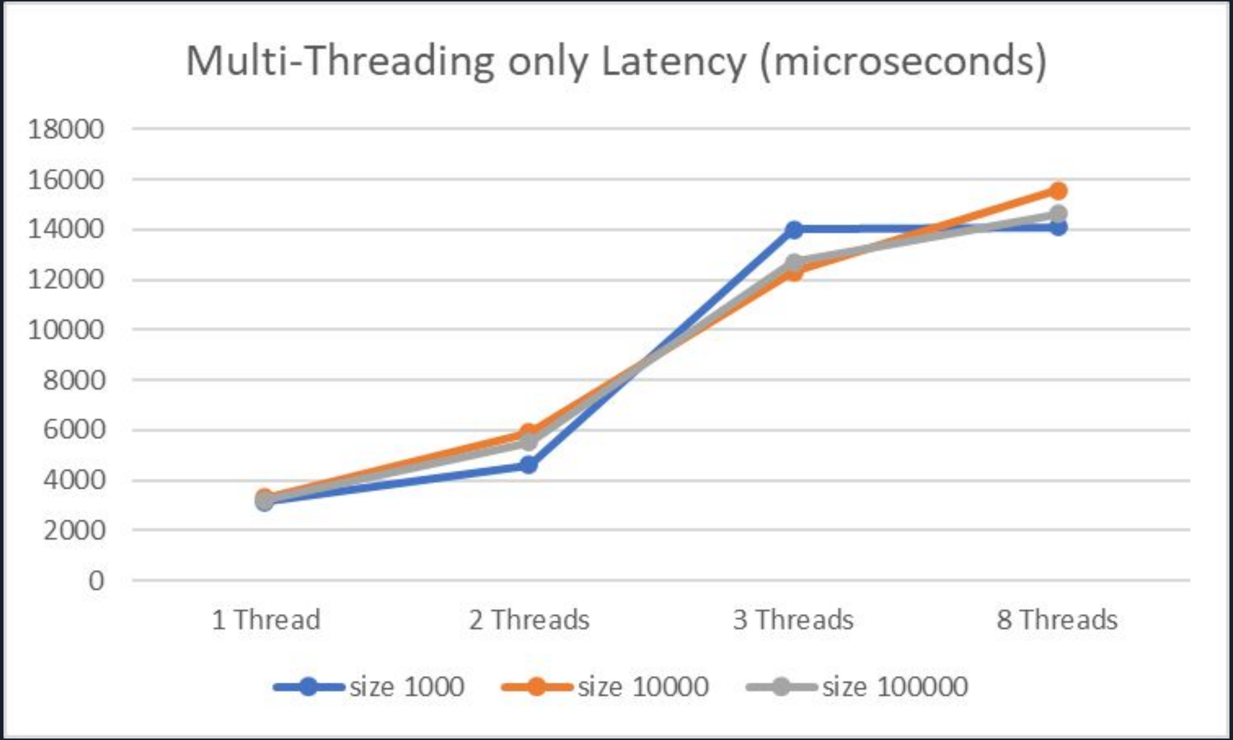
Each thread performs multiple search operations, and the latency and throughput are calculated and printed for each thread count.

Code

In our SIMD version:

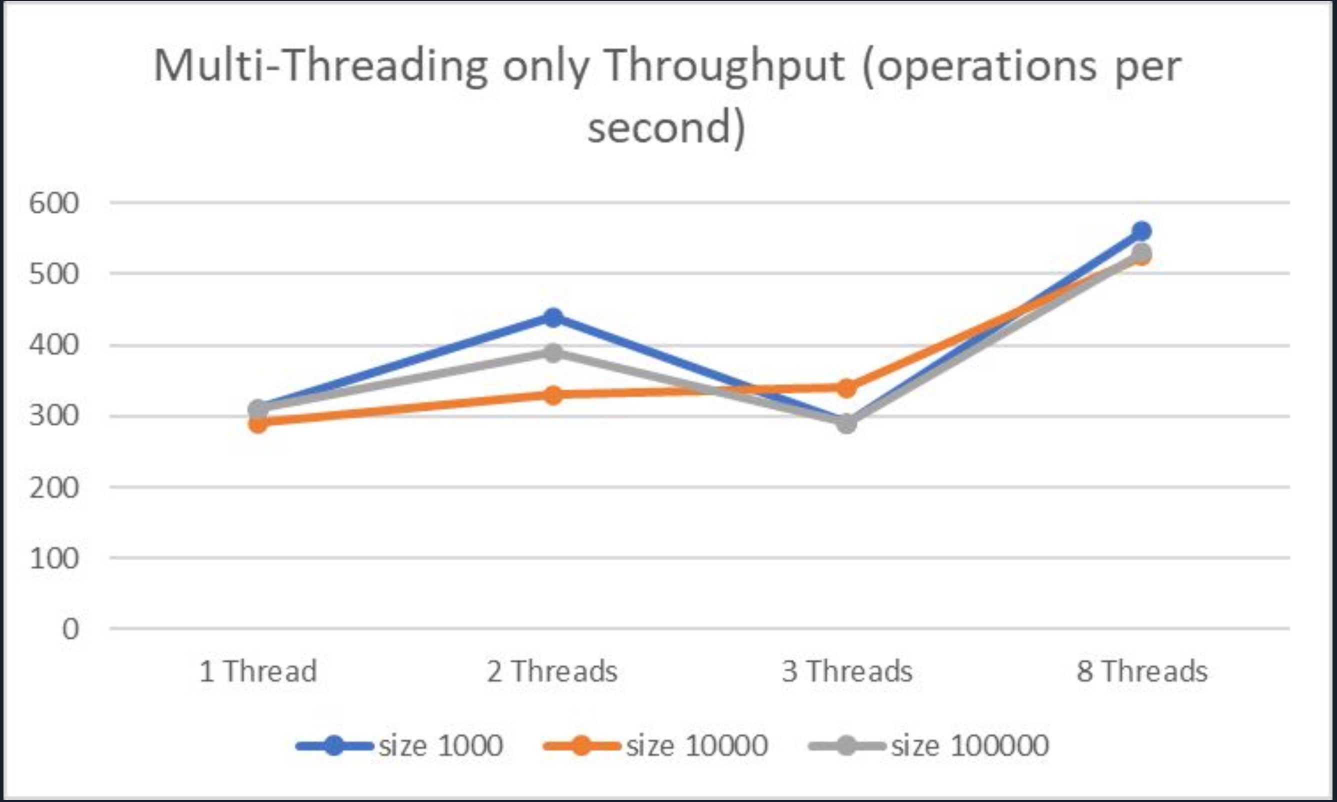
SIMD is used to search for eight keys simultaneously, potentially improving search performance.

Graphs

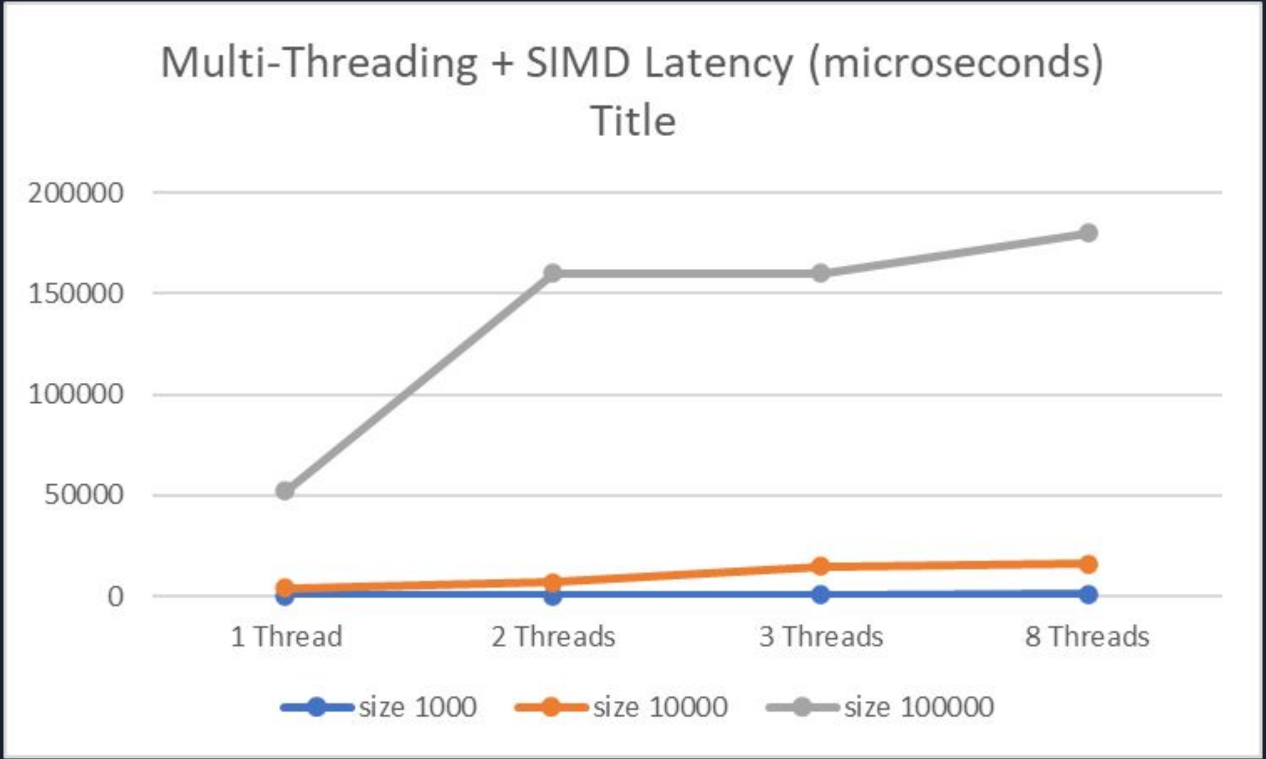


Multi-Threading only Latency (microseconds)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1 Thread
2 Threads
3 Threads
8 Threads
size 1000
size 10000
size 100000

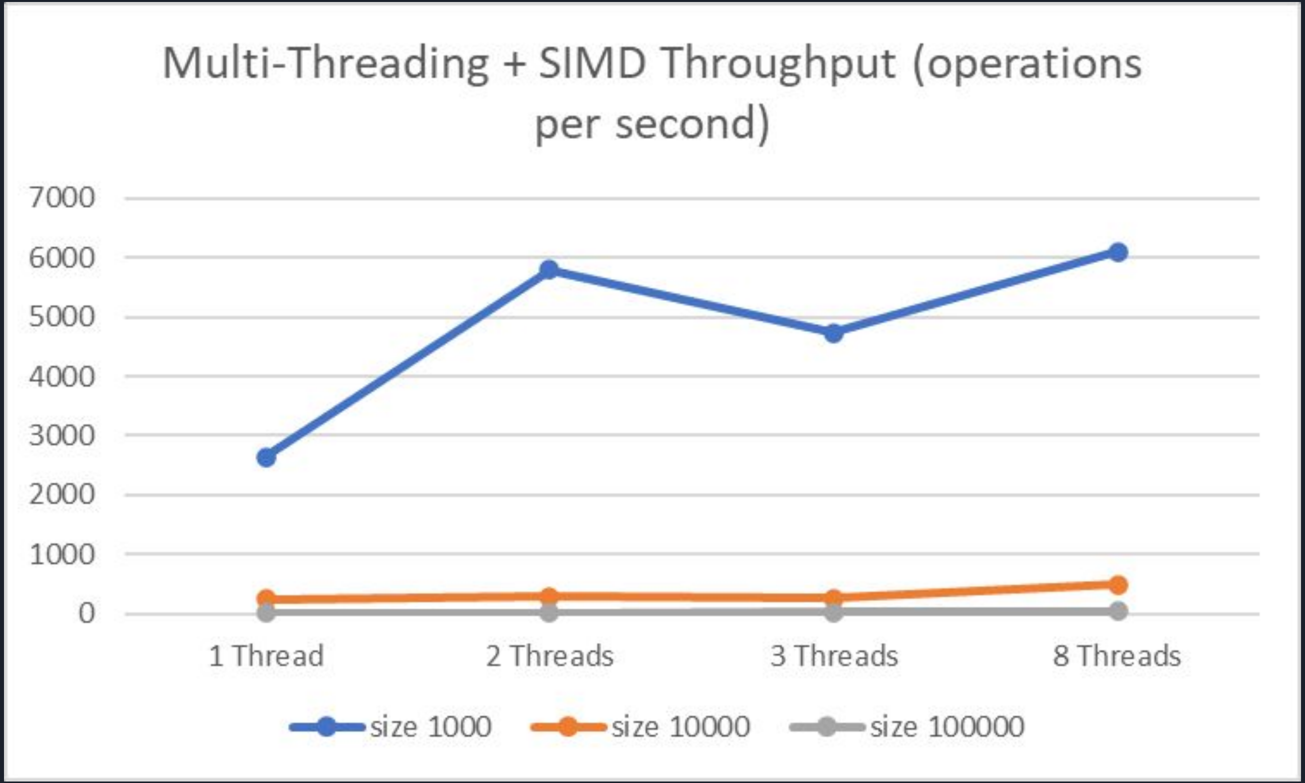
Graphs


Multi-Threading only Throughput (operations per second)
600
500
400
300
200
100
0
1 Thread
2 Threads
3 Threads
8 Threads
size 1000
size 10000
size 100000

Graphs


Multi-Threading + SIMD Latency (microseconds)
Title
200000
150000
100000
50000
0
1 Thread
2 Threads
3 Threads
8 Threads
size 1000
size 10000
size 100000

Graphs



Multi-Threading + SIMD Throughput (operations per second)
7000
6000
5000
4000
3000
2000
1000
0
1 Thread
2 Threads
3 Threads
8 Threads
size 1000
size 10000
size 100000

Big picture

- We can make repetitive processes faster using hash tables, lists and other structures that can easily run these processes in loops
- These structures can then use concurrency to really speed up the processes' execution time
- However, we have to be wary of things like blocking and random errors that can occur and make a lot of flags and really think about how to implement our memory structures when utilizing threads

What we learned

- Advantages of concurrency
- Disadvantages of concurrency
- Problems in compilation with concurrency
- Ways to resolve and debug concurrency
- Different types of concurrency
- Types of structures that support concurrency