747993_SHAANREHSI_A3

1. Implement the algorithm find_all() that returns a vector<T*> of pointers to all elements that meet the criteria.

   Using a vector<int> of 10000000:

ITERATION	Elements Meeting the criteria	Executed time microseconds
1.	2993700	101319
2.	2993700	99691
3.	2993700	93513

AVERAGE TIME: 98174

The find_all() function takes a const std::vector<T>& data as input and a T threshold. The function searches for elements in the input vector (data) that are greater than the given threshold.

Using a vector<string> of 1000000:

ITERATION	Elements Meeting the criteria	Executed time microseconds
1.	999990	30516
2.	999990	29634
3.	999990	27856

AVERAGE TIME: 29335

2. Build a 10-way parallel find_all(): given a vector, spawn 10 threads each running a serial find_all() on a 10th of the elements and combining their results.

   Measure its performance, excluding the thread creation. The easiest would be to make threads set up to take messages from a synchronized queue or from a future

Using a vector<int> of 10000000:

ITERATION	Elements Meeting the criteria	Executed time microseconds
1.	2993700	46064
2.	2993700	41363
3.	2993700	40580

AVERAGE TIME: 42669

Using a vector<string> of 1000000:

ITERATION	Elements Meeting the criteria	Executed time microseconds
1.	39294	6882
2.	39248	6661
3.	39180	6840

AVERAGE TIME: 6794

To build a 10-way parallel find_all(): I used the std::async function to spawn 10 threads, each running a serial find_all() on a tenth of the elements from the input vector. This allows the work to be divided among multiple threads, making the search process parallel.

To exclude thread creation time I used a synchronized queue using std::mutex and std::condition_variable to pass messages from the worker threads back to the main thread. This ensures that the main thread waits for all worker threads to complete their tasks before collecting their results.
To measure performance I used std::chrono to measure the execution time of the parallel find_all() algorithm, providing the execution time in microseconds.

EVALUATION:

Each piece of code I ran has a fairly similar execution time although there is some variation. This variation is due to factors like system load, thread scheduling, and background processes. Although I only ran the code three times, the consistency of the execution time can conclude that the parallel algorithm is performing well and relatively quick.