Project 2 - Thread Pool

CmpE 322, Operating Systems, Fall 2022

Bahadır Gezer - 2020400039

November 27, 2022

Implementation Details

This project spans three source files. Two of them `operations.cpp/.h` has the 10 operations implemented in it. These implementations assume that the input array is unsorted. The last file has the main function. `pthreads` are used inside the main function. The number of threads used depend on the second program argument. These threads are created to form a thread pool. Then the operations are fairly distributed to the available threads in this pool. I did not implement sub-array operations so the number of threads do not exceed 10. But other than that any number of threads is accepted. Timing is done through the monotonic clock using the `clock_gettime` function.

Threading Analysis

In this thread-pool example we can see that there are performance gains up until 4 to 5 threads. After that point however there are no more gains to be had. This might be because of the amount of parallelism the CPU can handle, or because of thread creation overhead, or because of the task distribution overhead. Whatever it might be the case we can see a lower bound in execution times when we have more than 5 threads.

It should also be pointed out that there is a massive performance gain when going from 1 thread to 2 or three threads. This increase is the initial gains from parallelism. These gains drop of as marginal returns from extra threads diminish.

Run Configuration

The `make` command will compile, assemble and run the program correctly. It uses 100.000 sized array and 10 threads by default. To run the program on your own write `./out/prog <array size> <number of threads>` to the terminal.

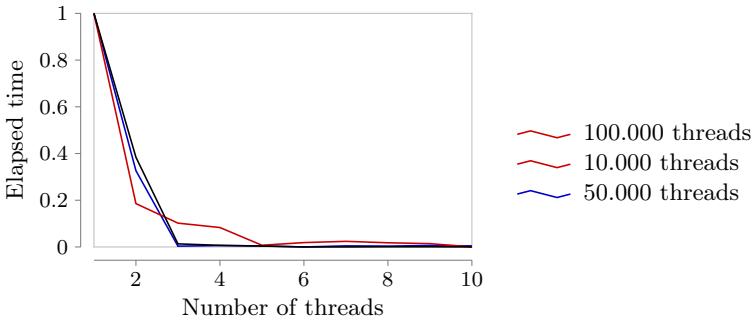

Figure 1: Normalized thread running times.