Coursework 2a: Threaded C++

System Programming (COMPSCI 4081)

1. Status

- Submission Option: 3 (An implementation that completely conforms to the full specification in Section 2 above).
- Compilation Status: It compiles without any errors or warnings.
- Bug Assessment: I am not aware of any bugs in my code.

2. Build, and sequential (i.e., original) & 1-thread runtimes

- Path where I am executing the program → **/users/level3/293710s9/SP/UofG_SP/Lab8_cw2/**
- Crawler is compiled in Lab8_cw2 folder by the Makefile with `make`

```
-bash-4.2$ pwd
/users/level3/293710s9/SP/UofG_SP/Lab8_cw2
-bash-4.2$ make
clang++ -Wall -Werror -std=c++17 -o dependencyDiscoverer dependencyDiscoverer.cpp -lpthread
-bash-4.2$
```

  - It can also be compiled, if preferred from the test folder with the command → `$ cd .. ; make ; cd test`
  - The sequential version is compiled with: `clang++ -Wall -Werror -std=c++17 -o sequential_fromMoodle sequential_fromMoodle.cpp -lpthread`
  - NOTE: The university servers might throw an error saying that C++17 is not available. You need to use a more recent version of Clang. To obtain it, run the following in the command shell on one of the stlinux servers (not ssh or sibu): `% source /usr/local/bin/clang9.setup`

- Time to run the sequential crawler on all .c, .l and .y files in the test directory:

```
-bash-4.2$ pwd
/users/level3/293710s9/SP/UofG_SP/Lab8_cw2
-bash-4.2$ make
clang++ -Wall -Werror -std=c++17 -o dependencyDiscoverer dependencyDiscoverer.cpp -lpthread
-bash-4.2$ export CRAWLER_THREADS=0
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.053s
user    0m0.007s
sys     0m0.018s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.046s
user    0m0.008s
sys     0m0.015s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.050s
user    0m0.010s
sys     0m0.014s
-bash-4.2$
```

  - The program run sequentially, like the code given to us on Moodle, with `CRAWLER_THREADS=0`.

- Time to run your threaded crawler (if you have one) with one thread:

```
-bash-4.2$ export CRAWLER_THREADS=1
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.052s
user    0m0.010s
sys     0m0.017s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.056s
user    0m0.012s
sys     0m0.013s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.052s
user    0m0.011s
sys     0m0.017s
-bash-4.2$
```

3. Runtime with Multiple Threads

a. Screenshots

Path where I am executing the program → **/users/level3/293710s9/SP/UofG_SP/Lab8_cw2**

Execution example with CRAWLER_THREADS=1:

```
-bash-4.2$ export CRAWLER_THREADS=1
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.052s
user    0m0.010s
sys     0m0.017s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.056s
user    0m0.012s
sys     0m0.013s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.052s
user    0m0.011s
sys     0m0.017s
```

Execution example with CRAWLER_THREADS=2:

```
-bash-4.2$ export CRAWLER_THREADS=2
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.042s
user    0m0.013s
sys     0m0.020s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.029s
user    0m0.010s
sys     0m0.017s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.029s
user    0m0.012s
sys     0m0.015s
```

Execution example with CRAWLER_THREADS=3:

```
-bash-4.2$ export CRAWLER_THREADS=3
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.025s
user    0m0.010s
sys     0m0.019s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.024s
user    0m0.015s
sys     0m0.013s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.024s
user    0m0.011s
sys     0m0.019s
```

Execution example with CRAWLER_THREADS=4:

```
-bash-4.2$ export CRAWLER_THREADS=4
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.023s
user    0m0.012s
sys     0m0.019s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.022s
user    0m0.009s
sys     0m0.021s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.022s
user    0m0.011s
sys     0m0.018s
```

Execution example with CRAWLER_THREADS=6:

```
-bash-4.2$ export CRAWLER_THREADS=6
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.021s
user    0m0.013s
sys     0m0.020s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.021s
user    0m0.014s
sys     0m0.024s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.020s
user    0m0.012s
sys     0m0.019s
```

Execution example with CRAWLER_THREADS=8:

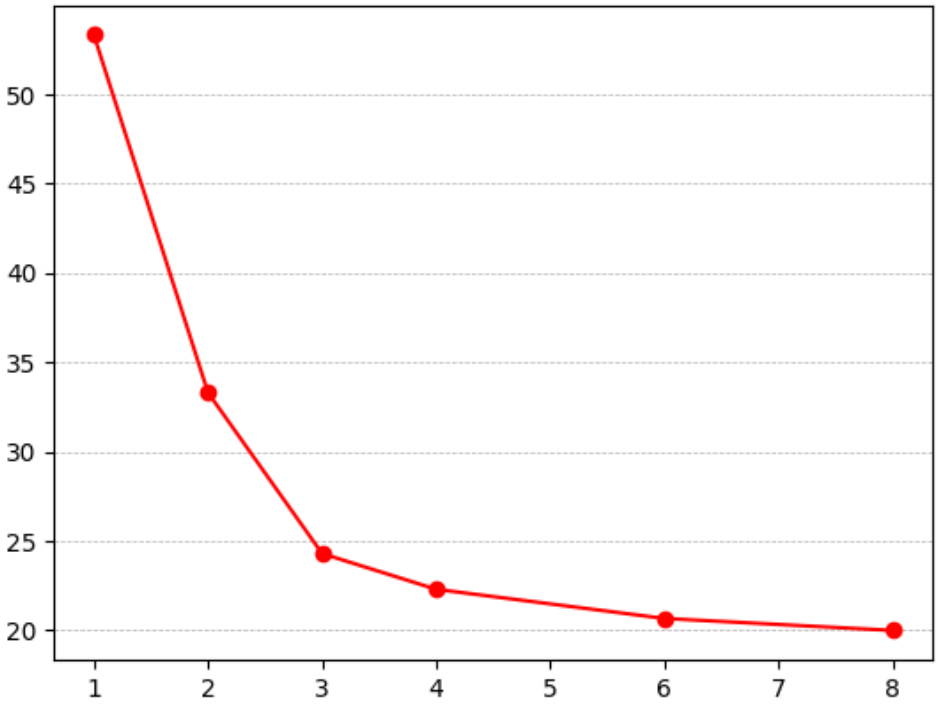
```
-bash-4.2$ export CRAWLER_THREADS=8
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.019s
user    0m0.019s
sys     0m0.015s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.021s
user    0m0.012s
sys     0m0.029s
-bash-4.2$ time ./dependencyDiscoverer -Itest test/*.c test/*.l test/*.y > temp

real    0m0.020s
user    0m0.010s
sys     0m0.028s
-bash-4.2$
```

b. Experiment

- Times to run the crawler with 1, 2, 3, 4, 6, and 8 threads on all .c, .l and .y files in the test directory

The next table includes the execution elapsed time of `./dependencyDiscover` in milliseconds with different number of threads

CRAWLER_ THREADS	1	2	3	4	6	8
Execution 1	52	42	25	23	21	19
Execution 2	56	29	24	22	21	21
Execution 3	52	29	24	22	20	20
Median	53.3	33.3	24.3	22.3	20.67	20



c. Discussion

The results show that as the number of threads increase, the execution time of the program is reduced. The main reason for this is that having multiple threads benefit from multi-core, so the different threads can run in parallel and concurrent with the workload being distributed among multiple cores.

The program has two sequential parts, before creating the threads and the printing part, where there is only one thread, which lasts approximately the same time for each execution. The only part that is optimize with the increase of threads is the one with the threads. With a higher number of threads, the work benefits of the parallelization and concurrency. Therefore, this part run faster, approximately the part with threads run x times faster than running it sequentially, being x the number of threads.

In the results we also obtained that the sequential version runs faster than the version with 1 thread. This is because creating threads requires creating the new threads and allocating memory, which penalise the execution time.