Report, 2386300t

I have provided a solution that appears to work with any number of threads (the extent I tested went up to 60 and it ran with no error), so is a multithreaded solution. The solution compiles on the university servers with no issue and when run will match the output file with any number of threads set (up to 60 at least). The main changes made to the code were the addition of the thread safe data types, and step four of the main was altered to use threads, past this, the only changes were to method calls on the new data types.

Screenshots:

```
-bash-4.2$ pwd
/users/level3/2386300t/coursework2/ccoursework2/test
```

```
-bash-4.2$ make
clang++ -Wall -Werror -std=c++17 -o dependencyDiscoverer dependencyDiscoverer.cpp -lpthre
ad
```

With one thread:

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

real    0m0.085s
user    0m0.017s
sys     0m0.014s
```

With two threads:

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

real    0m0.046s
user    0m0.009s
sys     0m0.023s
```

With three threads:

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

real    0m0.023s
user    0m0.014s
sys     0m0.012s
```

With four threads:

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

real    0m0.025s
user    0m0.012s
sys     0m0.020s
```

With six threads:

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

real    0m0.029s
user    0m0.011s
sys     0m0.027s
```

With eight threads:

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

real    0m0.020s
user    0m0.017s
sys     0m0.019s
```

Experiment

CRAWLER_ THREADS	1	2	3	4	6	8
	Elapsed Time	Elapsed Time	Elapsed Time	Elapsed Time	Elapsed Time	Elapsed Time
Execution 1	0.065	0.031	0.024	0.029	0.020	0.020
Execution 2	0.051	0.029	0.025	0.022	0.020	0.020
Execution 3	0.043	0.030	0.024	0.021	0.020	0.018
Median					0.020	

Would appear that, at least in this range on 1 to 8, the eight threads give the best result. However, I repeated the steps with a much larger thread count of 50 and found that the time was less than that of the eight threads, so the trend doesn't appear to continue the same way outside of the range. As you increase the threads, it also appears to decrease the variability of times, however a larger sample size would really be needed to say that for certain.

The results here seem to say in general that a greater number of cores results in both a lower variability and a lower execution time.

(I don't really see the need for section 2, as both the one thread and the sequential – which in itself is more or less the same as the one thread in my implementation – are covered above, if that stuff was required, using the one thread results would almost certainly give the same sort of time frame)