Advanced Programming - Assessed Exercise 2

Report

Andrei-Mihai Nicolae
2147392n

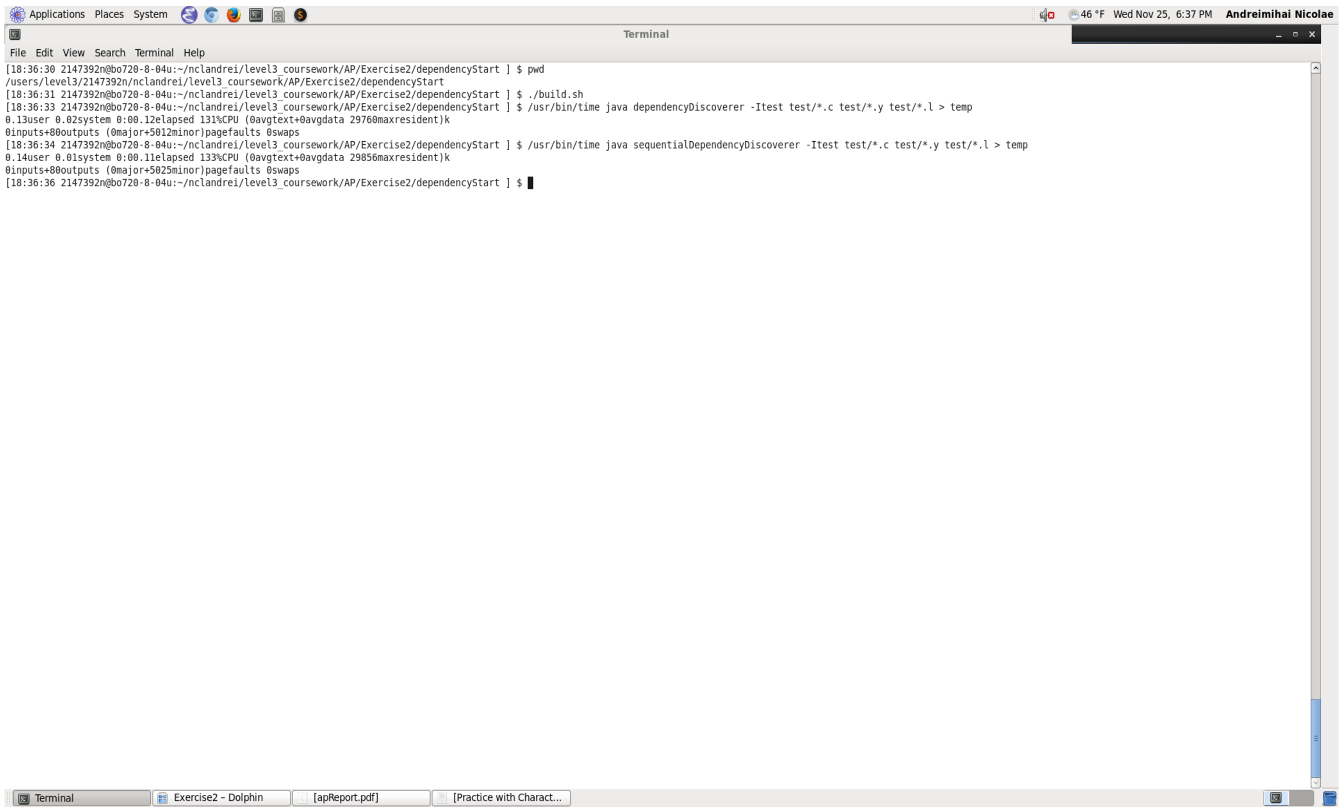
November 26, 2015

1 Submission status

The dependency discoverer is fully functional and very optimized. I have created the multi-threading variant and everything works perfectly. I made sure that the main thread knows exactly when to stop all working threads without any busy waiting (such that all of them work until the end and they are not killed prematurely), I have implemented a small feature (using CyclicBarrier) that allows the program to make all worker threads start at approximately the same time (such that they don't get created while the work process has already begun) and there is no chance of finding deadlocks/livelocks/starvation/spurious wakeups.
I have not described my algorithm inside here as it is well documented in terms of comments inside the file.
I have tested with both the "test" folder and an unseen folder and everything works perfectly.

Summing up, I consider that my implementation has fulfilled all the requirements for the multi-threading solution in the handout.

2 Build and Sequential and 1 Thread runtimes


Applications Places System
46 °F Wed Nov 25, 6:37 PM Andreimihai Nicolae
Terminal
File Edit View Search Terminal Help
Terminal
Exercise2 - Dolphin
[apReport.pdf]
[Practice with Charact...

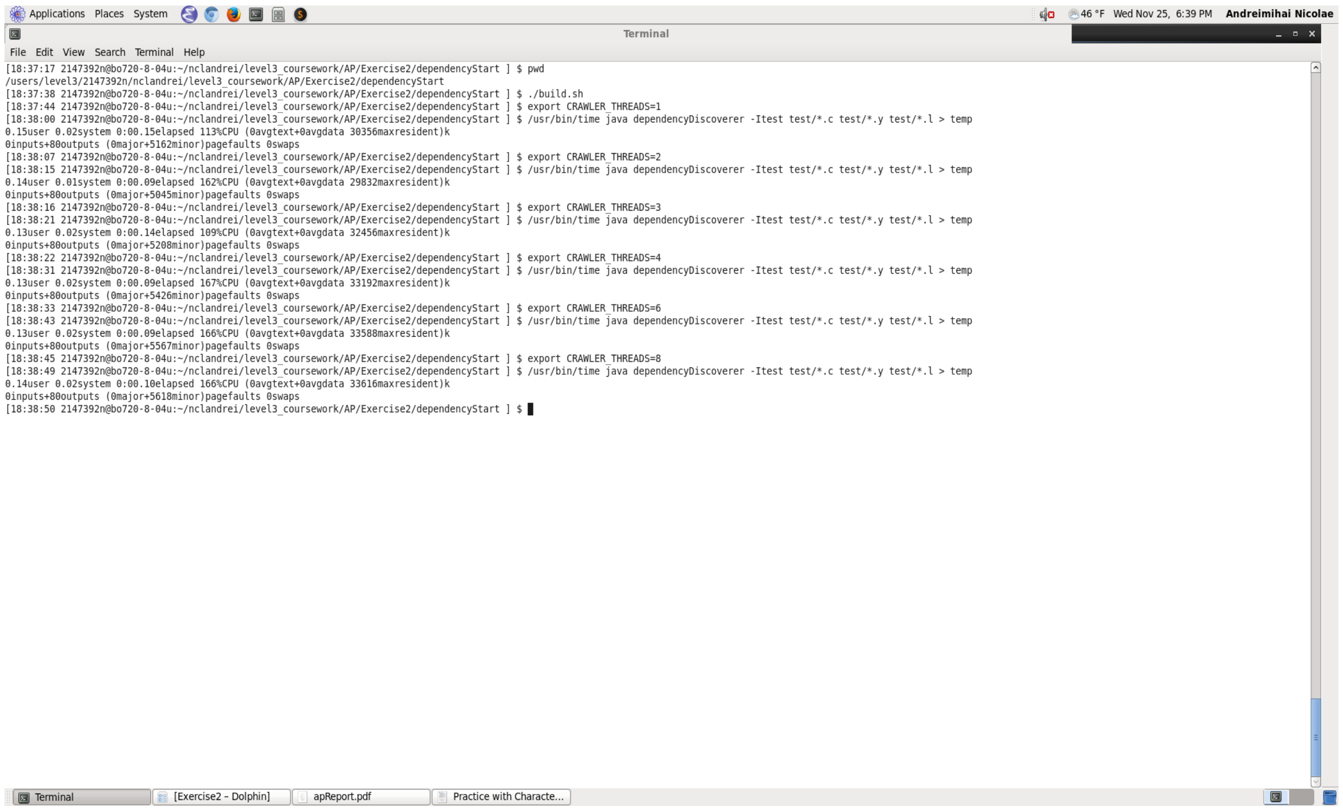
```
[18:36:30 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ pwd
/users/level3/2147392n/nclandrei/level3_coursework/AP/Exercise2/dependencyStart
[18:36:31 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ ./build.sh
[18:36:33 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.13user 0.02system 0:00.12elapsed 131%CPU (0avgtext+0avgdata 29760maxresident)k
0inputs+80outputs (0major+5012minor)pagefaults 0swaps
[18:36:34 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java sequentialDependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.14user 0.01system 0:00.11elapsed 133%CPU (0avgtext+0avgdata 29856maxresident)k
0inputs+80outputs (0major+5025minor)pagefaults 0swaps
[18:36:36 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $
```

As you can see here, the time elapsed is the same (varies by 1 millisecond on rare occasions) and the CPU load is approximately the same (again, depends on the load of the machine, what threads is the CPU actually running at the moment etc.).

Therefore, our single-threaded implementation works very similar to the sequential version.

3 Runtime with Multiple Threads

3.1 Screenshot



```
[18:37:17 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ pwd
/users/level3/2147392n/nclandrei/level3_coursework/AP/Exercise2/dependencyStart
[18:37:38 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ ./build.sh
[18:37:44 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=1
[18:38:00 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.15user 0.02system 0:00.15elapsed 113%CPU (0avgtext+0avgdata 30356maxresident)k
0inputs+80outputs (0major+5162minor)pagefaults 0swaps
[18:38:07 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=2
[18:38:15 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.14user 0.01system 0:00.09elapsed 162%CPU (0avgtext+0avgdata 29832maxresident)k
0inputs+80outputs (0major+5045minor)pagefaults 0swaps
[18:38:16 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=3
[18:38:21 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.13user 0.02system 0:00.14elapsed 109%CPU (0avgtext+0avgdata 32456maxresident)k
0inputs+80outputs (0major+5208minor)pagefaults 0swaps
[18:38:22 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=4
[18:38:31 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.13user 0.02system 0:00.09elapsed 167%CPU (0avgtext+0avgdata 33192maxresident)k
0inputs+80outputs (0major+5426minor)pagefaults 0swaps
[18:38:33 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=6
[18:38:43 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.13user 0.02system 0:00.09elapsed 166%CPU (0avgtext+0avgdata 33588maxresident)k
0inputs+80outputs (0major+5567minor)pagefaults 0swaps
[18:38:45 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ export CRAWLER_THREADS=8
[18:38:49 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ /usr/bin/time java dependencyDiscoverer -Itest test/*.c test/*.y test/*.l > temp
0.14user 0.02system 0:00.10elapsed 166%CPU (0avgtext+0avgdata 33616maxresident)k
0inputs+80outputs (0major+5618minor)pagefaults 0swaps
[18:38:50 2147392n@bo720-8-04u:~/nclandrei/level3_coursework/AP/Exercise2/dependencyStart ] $ █
```

3.2 Experiment

Table 1: Experimenting with different crawler threads

CRAWLER_ THREADS	1		2		3		4		6		8	
	El. Time	Core Util.	El. Time	Core Util.	El. Time	Core Util.	El. Time	Core Util.	El. Time	Core Util.	El. Time	Core Util.
Execution 1	0:00.13	113%	0:00.10	138%	0:00.10	133%	0:00.9	137%	0:00.11	145%	0:00.11	148%
Execution 2	0:00.11	129%	0:00.9	140%	0:00.10	142%	0:00.8	172%	0:00.9	161%	0:00.9	150%
Execution 3	0:00.12	116%	0:00.10	130%	0:00.11	148%	0:00.11	139%	0:00.9	153%	0:00.10	146%
Median	0:00.12	116%	0:00.10	138%	0:00.10	142%	0:00.9	139%	0:00.9	153%	0:00.8	148%

3.3 Discussion

The results from experimenting with different number of threads on different loads of the machine has proven to be quite interesting and insightful. One might think that a multi-threaded program will generally have a ”sweet spot” number of threads for minimum CPU load. Is that true? Not quite, mainly because even if we have an infinite number of waiting threads, the waiting process does not take any CPU time at all. Therefore, our time is mainly consumed during the I/O operations (e.g. when trying to open/read files).

Coming back to the results, I have noticed a curve in terms of time (going downwards when the number of threads was increased). However, even though the CPU utilization had variations, these were not relevant. The median on all of my runs (approx. 15-20 on each number of threads given above in the table) was usually the same, thus waiting proves to be very efficient and does not increase the load. This changes, however, when we have a huge number of files: I have created a script that spawned hundreds of .c, .y and .l files with random includes. When ran with a small number of threads (like the ones above) and a much larger one, the performance was, indeed, drastically increased. Therefore, the link I have observed was between the number of files to be processed/tasks to be completed and the number of threads: while we continue to add a lot of tasks, then, in order to have improved efficiency, we should also increase the number of threads to maximize our CPUs power.

In conclusion, every programmer designing an application that might work in a multi-threaded environment should test, if he/she had the possibility, the optimal number of threads for his application and thus get the right balance between tasks to be done and time efficiency.