CS3210 Assignment 1

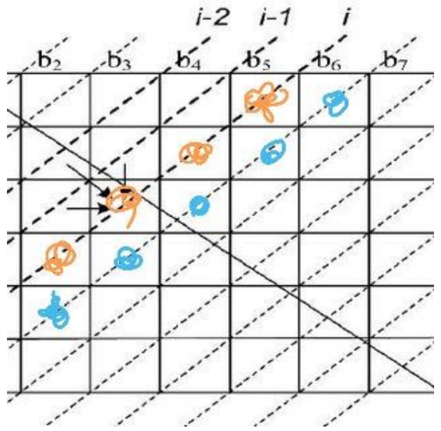
Bobbie Soedirgo (A0181001A), Daniel Alfred Widjaja (A0184588J)

September 13, 2020

1 Program Design

The dynamic programming of Longest Common Subsequence (LCS) is already defined as

$$LCS[i,j] = \begin{cases} 0, i = 0 \vee j = 0 \\ LCS[i-1, j-1] + 1, x_i = y_i \\ max(LCS[i-1, j], LCS[i, j-1]), otherwise \end{cases}$$



Notice that we could do the computation on all the orange squares above before the blue squares. This allows us to create barrier later when doing parallelism. The barrier prevent any orange square is uncomputed when we start doing the blue squares that needed the value in the orange square.

2 Parallelization Strategy

Using semaphores, paralellization is done using one mutex (i.e. semaphore(1)) and two barriers (semaphore(0)). The mutex is used to guarantee atomic increment of the count variable. The choice to use two barriers is an implementation detail: it allows us to alternate between them so one process doesn't break through the next barrier while the current processes are trying to go through the current barrier.

When implemented using mutex, it allows us to use conditional variable (CV), which makes it only need 2 mutex and 1 conditional variable for the whole implementation. 1

mutex for ensure the done count is mutually exclusive. Done count contains the number of done threads in the given barrier. Every time the one thread is done for the given barrier, it will have a CV which will awakes once the done counter has reached the number of threads (i.e. when every threads have done their job for the given barrier).

3 Reproducing Results

The machines used were soctf-pdc-019 for the server machine and soctf-pdc-021 for the desktop machine.

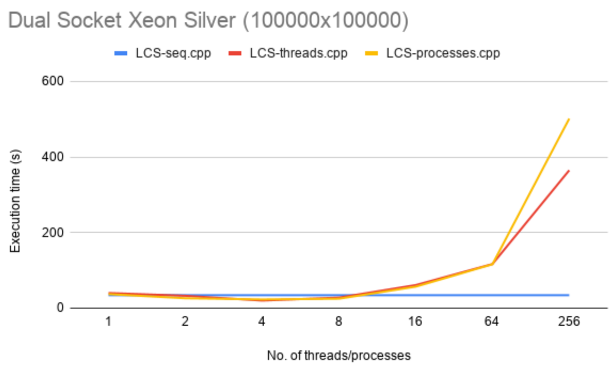
To compile, e.g. for LCS-seq.cpp, run:

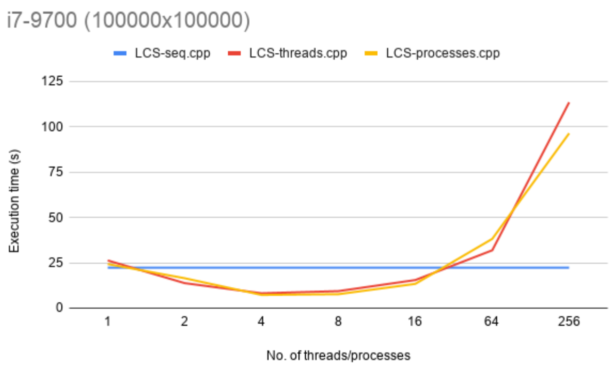
```
g++ -O3 -o lcs-seq LCS-seq.cpp
```

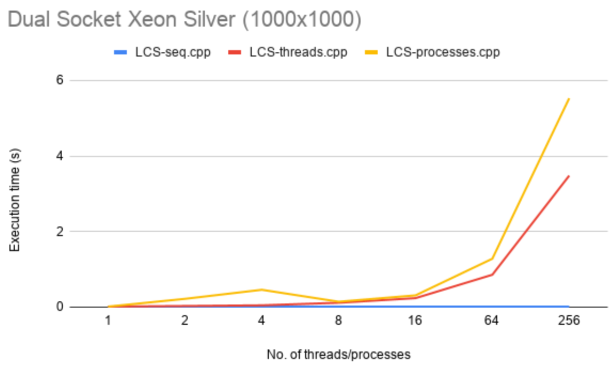
To change the no. of threads/processes, change N_THREADS for LCS-threads.cpp or N_PROCS for LCS-processes.cpp. To measure execution time, run e.g.:

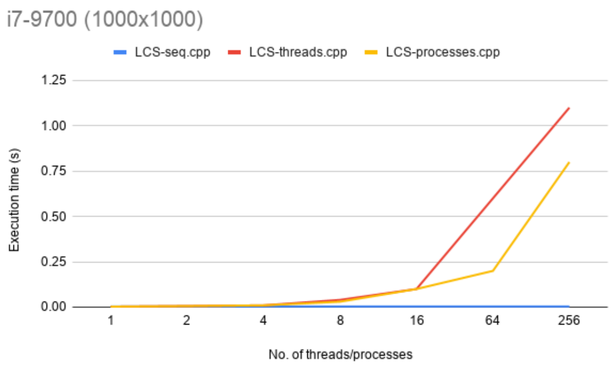
```
perf stat -e duration_time – ./lcs-seq DNA-10-1.in DNA-10-2.in
```

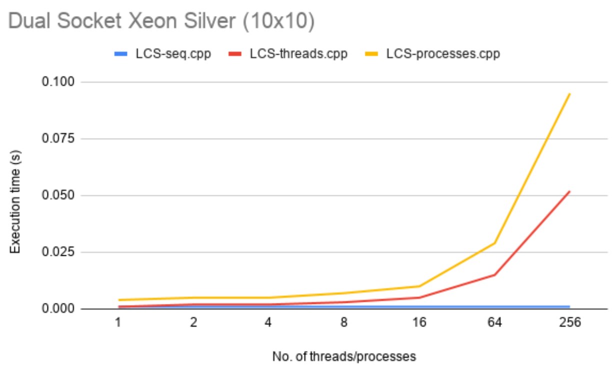
4 Measurement Results

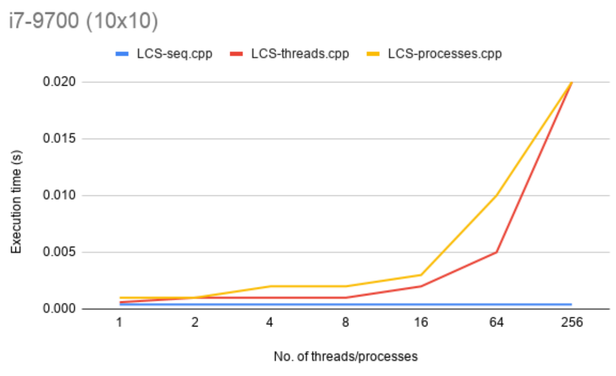












In our experiment, the clearest sign of performance improvement is between 2 to 16 threads/processes on the i7-9700 machine for the largest problem size (100000x100000). We suspected more performance gains on the Xeon machine, but in hindsight they don't perform as well on 2 to 16 threads/processes compared to the i7 machine. Further parallelization gives more overhead than the performance gains.

5 Conclusions

Parallelization increases performance in some cases, but sometimes the overhead nullifies the performance gains.