George Koubbe

First experimental results on RADICAL-EnTK

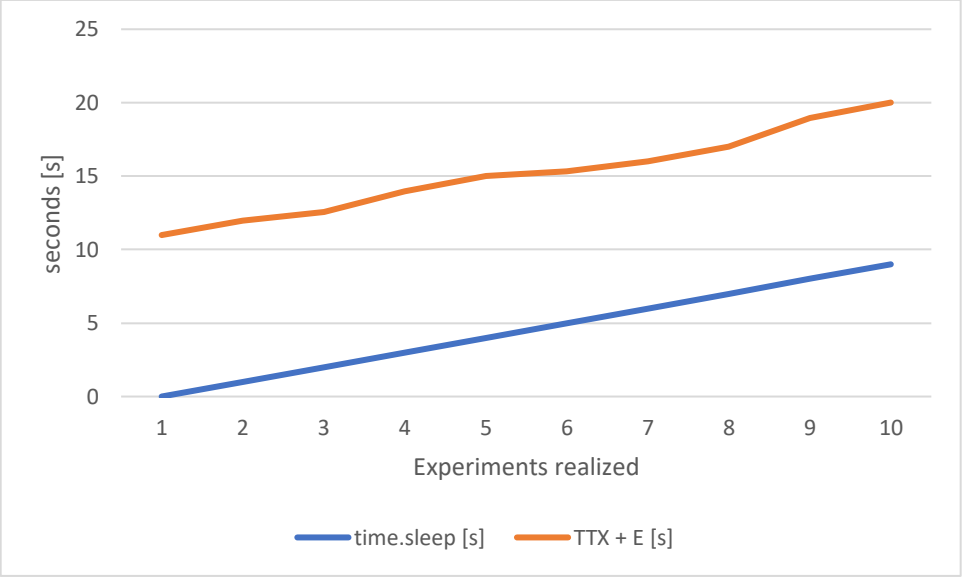
This work analyzes the TTX, or Total Time of Execution, of the following cases ran on Bridges supercomputer:

— 128 tasks concurrently, where each task is 1 core
— 8 tasks where each task is 16 cores
— 16 concurrent batches of 8 tasks (each of 1 core), but where in each batch each task runs sequentially, i.e., one after the other.

The tasks were of arbitrarily short duration.

To start, I wrote a Python script that all it did was to wait idle for one second (doNothing.py). I plotted a comparative table on increasing waiting times from zero to nine seconds, with one second intervals. I did this for:

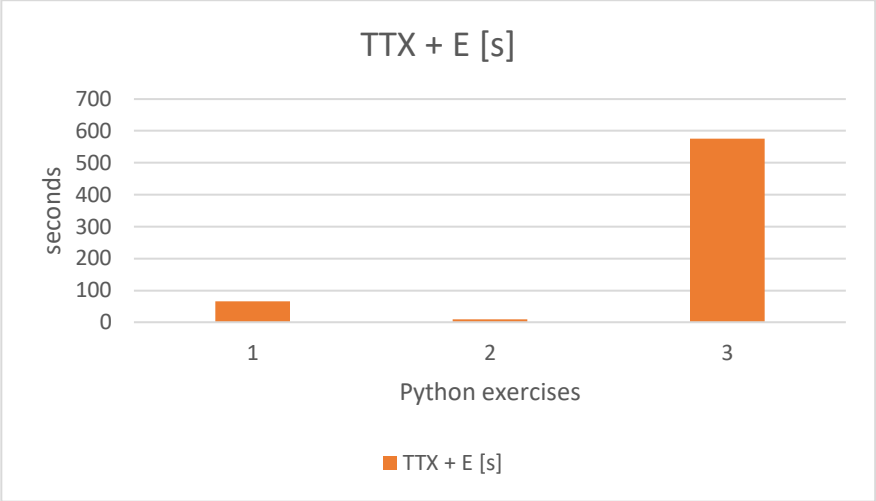
Localhost: 128 tasks concurrently, where each task is 1 core



As we can see, there is an approximate linear increase in the TTX, following the interval increase accordingly. This is what is expected, since all the tasks are running concurrently on all the cores, so the TTX growth will be that of the concurrently executed tasks.

xsede.bridges

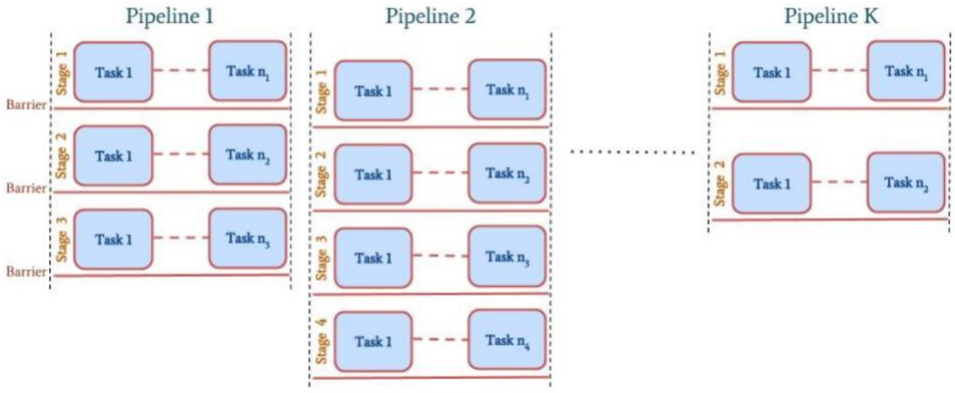
For the next stage, I compared the three suggested exercises using Bridges supercomputer. This time, I wrote a piece of code that runs in a serial fashion (SerialCode.py) and the same code that runs in a multiprocessing fashion (MultiprocessingCode.py), with 16 processes.



As we can see, for exercise 1 I ran all tasks concurrently on all 128 cores and get a TTX + E of about 65 seconds, which I can say is the approximate time to run 1 task on a single core at any given time.

For exercise 2, I obtained about 9 seconds which is approximately 7 times faster. This is what is expected, since now we are running the same piece of code on a multiprocessing fashion with 16 processes, thus the same task requires 16 cores.

Finally, I compared exactly the same code used for exercise 1 with exercise 3. This time, I ran 16 concurrent batches, but inside each batch, every task ran sequentially for a total of 8 tasks per batch. It is something like the following figure, where pipeline is equivalent to batch, and every stage has 1 task for a total of 8 stages.



I expected something like the TTX of exercise 1 multiplied by 8, given the sequential fashion, plus an overhead E from EnTK. Hence, I expected around 520 seconds + some E. What I observed was around 570 seconds. It matches my expectations.