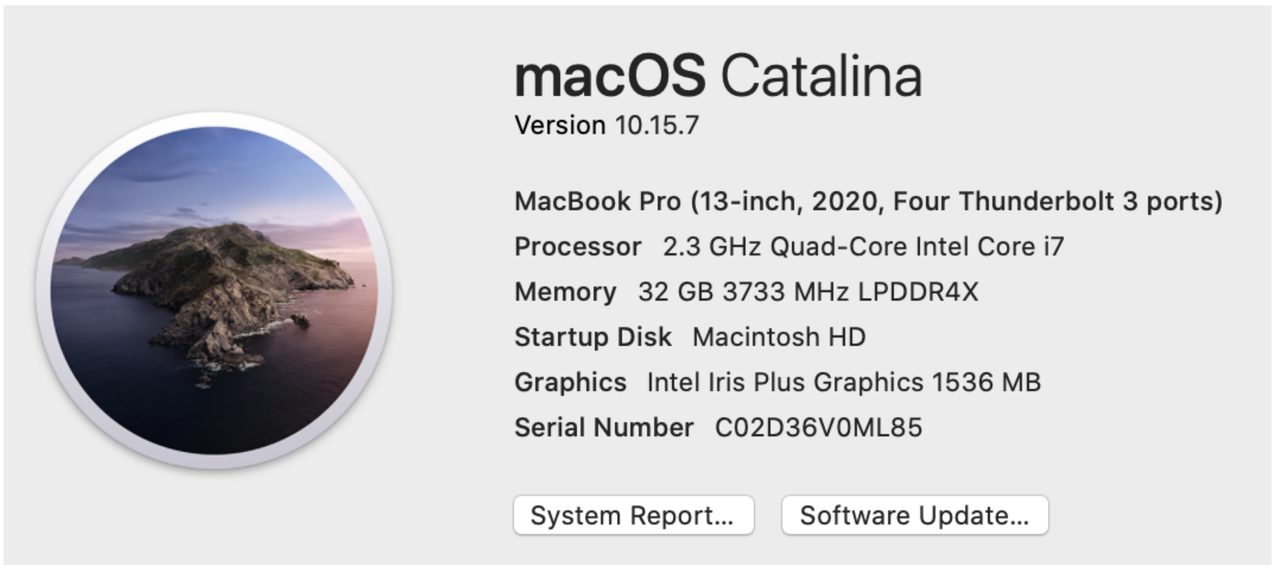
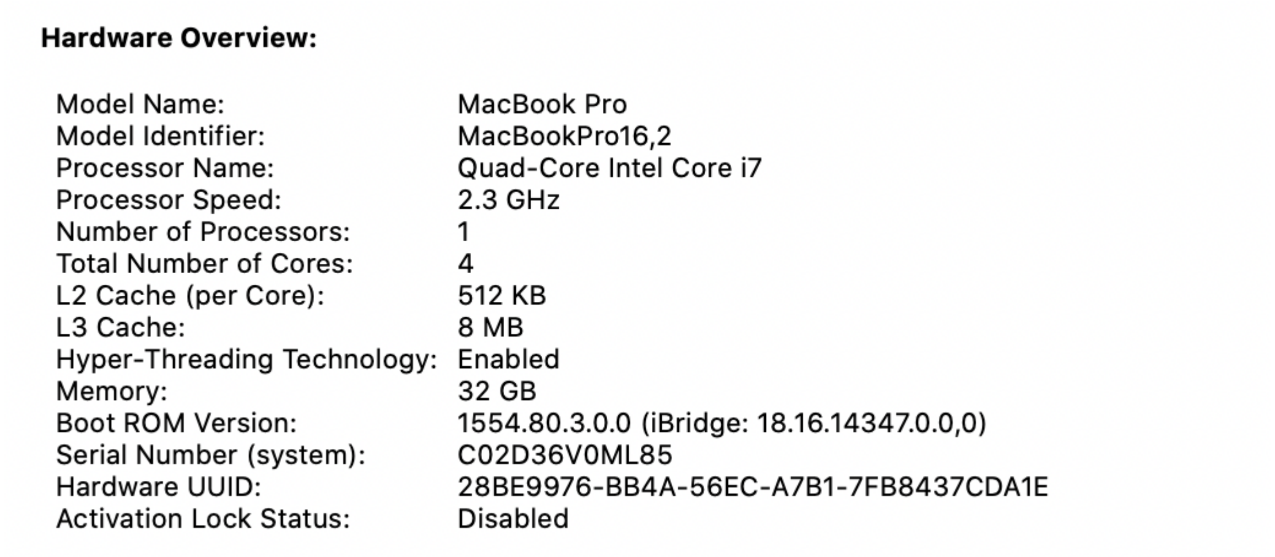
Spec:



macOS Catalina
Version 10.15.7

MacBook Pro (13-inch, 2020, Four Thunderbolt 3 ports)

Processor	2.3 GHz Quad-Core Intel Core i7
Memory	32 GB 3733 MHz LPDDR4X
Startup Disk	Macintosh HD
Graphics	Intel Iris Plus Graphics 1536 MB
Serial Number	C02D36V0ML85



Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro16,2
Processor Name:	Quad-Core Intel Core i7
Processor Speed:	2.3 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	512 KB
L3 Cache:	8 MB
Hyper-Threading Technology:	Enabled
Memory:	32 GB
Boot ROM Version:	1554.80.3.0.0 (iBridge: 18.16.14347.0.0,0)
Serial Number (system):	C02D36V0ML85
Hardware UUID:	28BE9976-BB4A-56EC-A7B1-7FB8437CDA1E
Activation Lock Status:	Disabled

- Spec of system:
  - Model: MacBook Pro 13 Inch
  - Number of CPUs: 1 Quad-Core 2.3GHZ Intel Core i7 CPU
  - Number of Core per CPU: 4 cores
  - Clock Rate: 2.3GHZ
  - Cache Memory: 512 KB L2 Cache (per Core); 8MB L3 Cache
  - Main Memory: 32 GB 3733 MHz LPDDR4X
- Cluster: N/A

- Operating System: Mac OS Catalina Version 10.15.7
- Compiler: Python 2.7.16 (default, Jun 5 2020, 22:59:21)
- Libraries: `multiprocessing` , `time` , `matplotlib` and `numpy` imported by Python script
- Others: N/A

1.2. How Much Faster? (10 points)

In `P12.py`, the `burnTime` has been changed to simply sleep for a parameterized amount of time.

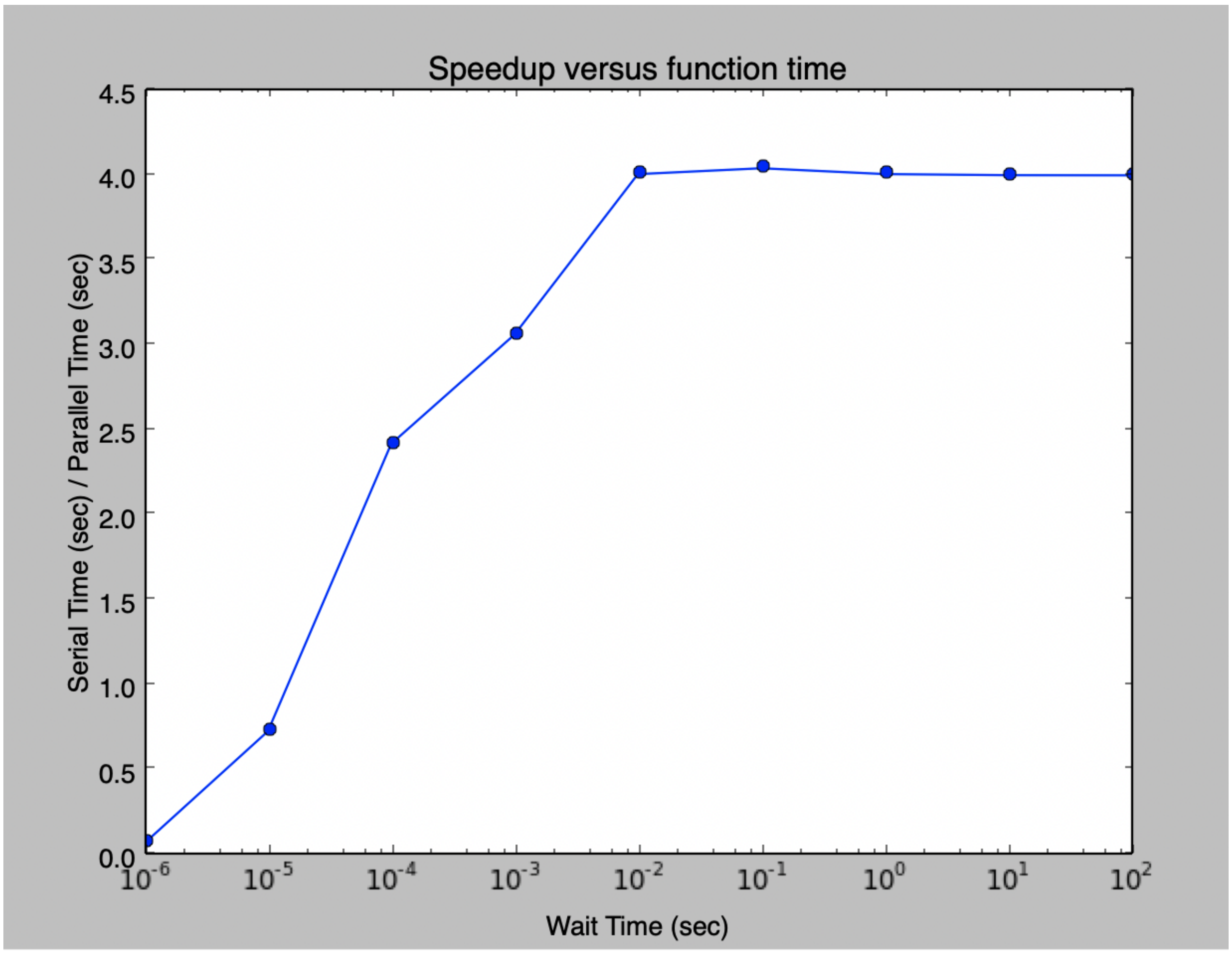
1. Using the `Pool.map` functionality, call `burnTime` 16 times in parallel using 4 processes and 16 times in serial using a standard loop. Use `time.time()` to determine how many seconds each takes and use various sleep times (ranging from 10^{-6} to 100 seconds) for each timing.
2. Plot the ratio of serial to parallel execution time against sleep time.
3. Try to explain the trend you observe.
4. Is it possible that a parallel program could take longer than it's serial version? If so, under what conditions does this occur?

Submission
- `P12.pdf`: Answers (including the plot) and discussion
- `P12.py`: Source code

> By Jiahui Tang

Q1. see P12.py

Q2. The plot of ratio of serial to parallel execution time against sleep time is attached below.



Q3.

We could see that ratio of `Serial Time/Parallel Time` increases dastically at the beginning as wait time goes from `1e-6 to 1e-2` sec. When wait time is extremely small, the ratio is less than one, means parallel time takes longer than serial time. This could resulted from overheads in communication, sync or load imbalance for using parallelism and multiprocessors, which may take longer than than a simple standard loop for serial code. After `1e−5` , the ratio is larger than 1, meaning parallelism takes effect and takes less amount of time than serial code.

And we could also observe that the ratio of `Serial Time/Parallel Time` approaches maximum ratio of 4 at around `1e−2` sec and the speedup saturated at 4 all the way till `100` sec. This is because we uses 4 processors in our multiprocessing thread pool. The maximum theortical parallel execution speed-up is thus $\frac{T(n,1)}{T(n,4)}$, which is 4 times speed up at maximum. Thus the ratio incremental slows down and stablize around 4 after a while.

Q4.

Yes it is possible that a parallel program takes longer time than its serial version. As we observe from the above diagram, under the condition that the `task takes small amount of time and is short` , the ratio < 1, meaning the parallel program takes longer than than serial version. This could due to the overheads in communication, sync or load imbalance for using parallelism and multiprocessors, allocating resources to processors and threads may take longer than than a simple standard loop for serial code.

Another example for taking longer time than serial is a `heavily coupled program that have dependencies` between different jobs, such as program involves race conditions. It requires locks for parallelism and waiting for previous process to finish, which may be ineffective comparing to using serial version. Its like a sudo parallism as the architecture for multiprocessor are not fully utilized in this case.