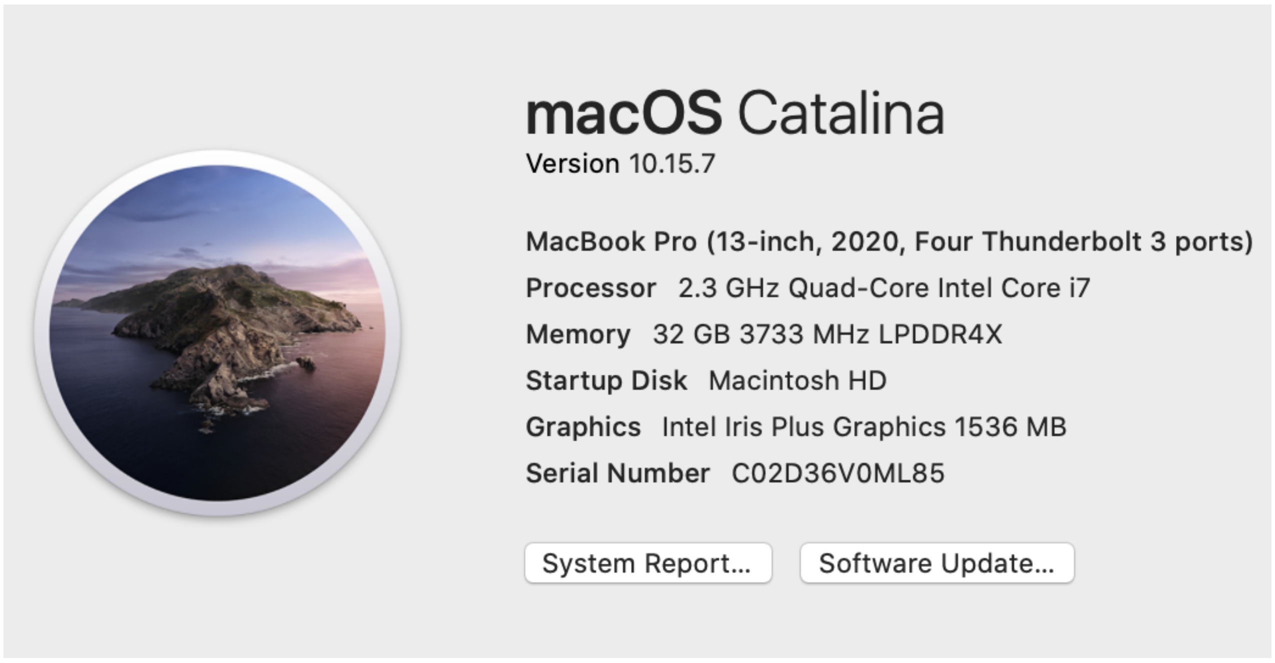
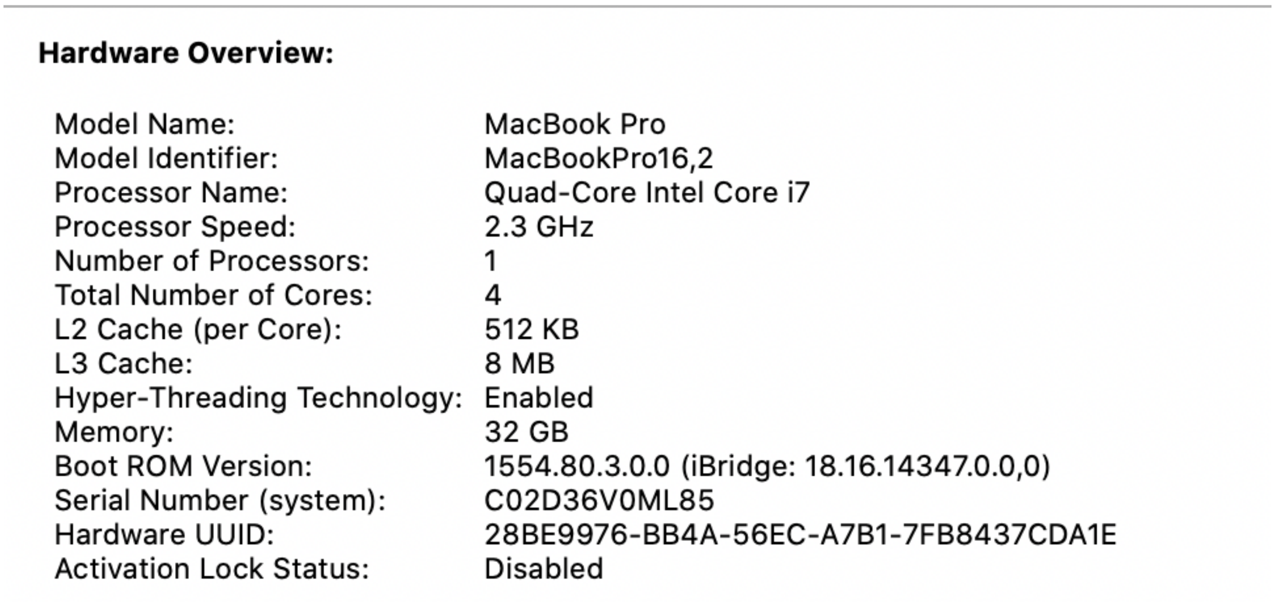
> By Jiahui Tang

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: Apple clang version 12.0.0 (gcc)

```
gcc -v
```

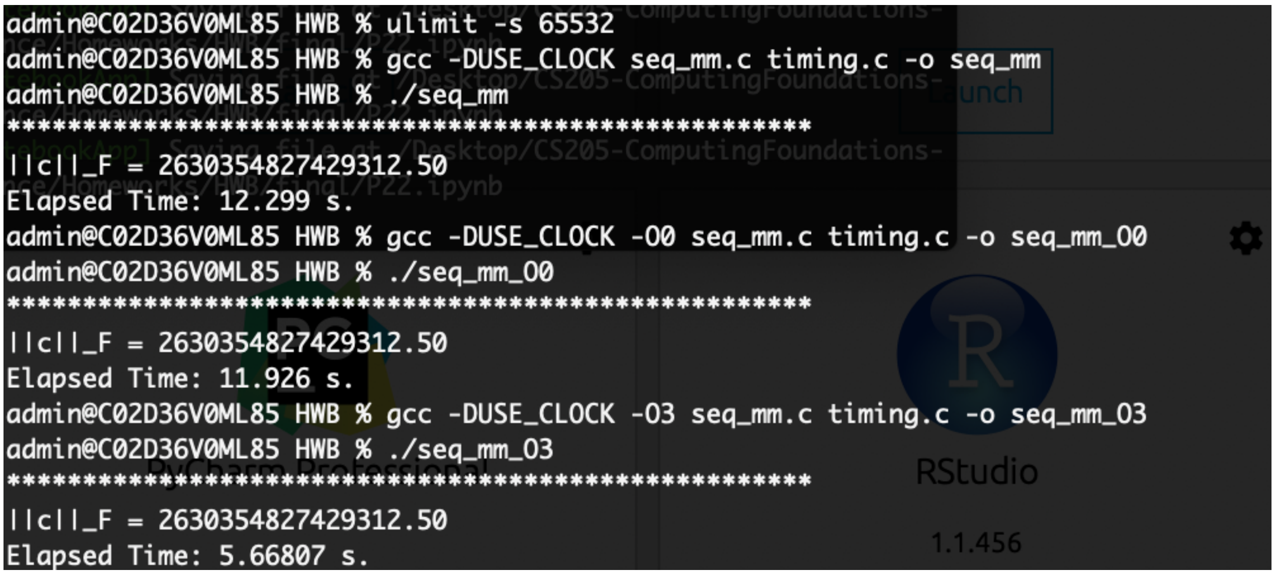
```
 Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-
gxx-include-
dir=/Library/Developer/CommandLineTools/SDKs/MacOSX10.15.sdk/usr/include/c++/4

Apple clang version 12.0.0 (clang-1200.0.32.29)
Target: x86_64-apple-darwin19.6.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

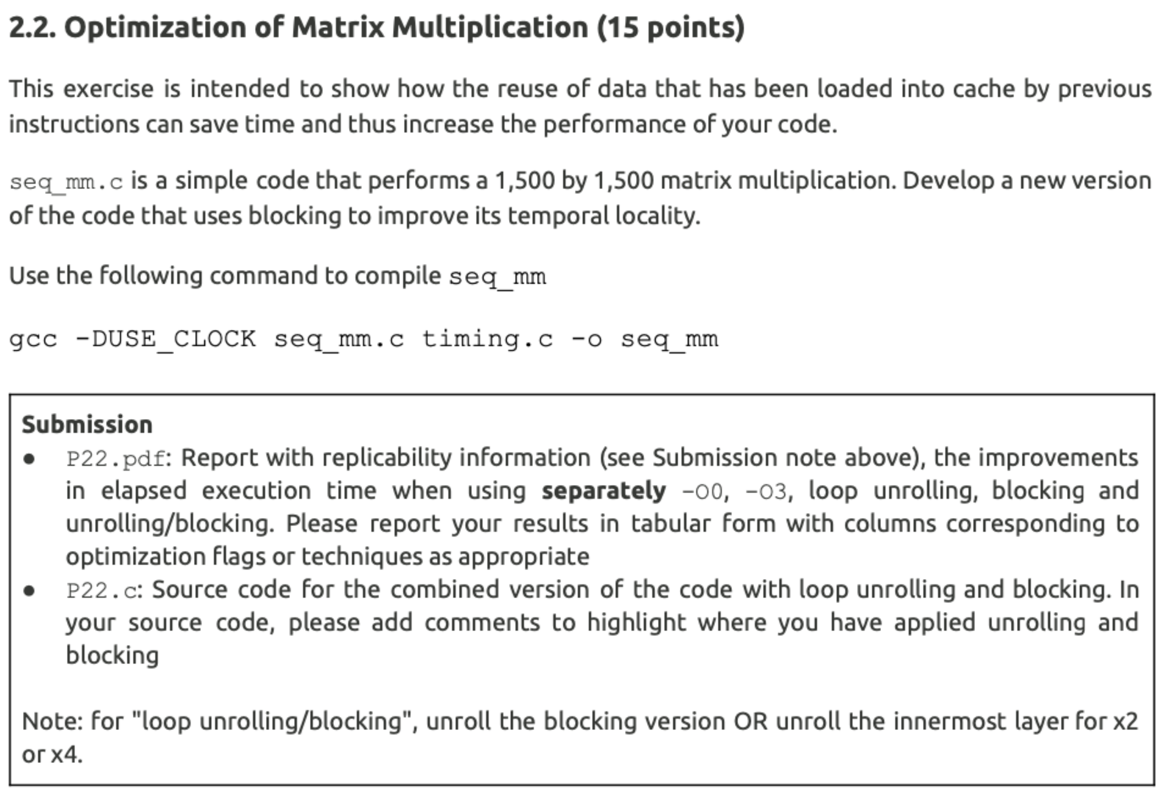
- Libraries: N/A
- Others: see below

Reproducibility Note:

For reproducibility, I run `ulimit -s 65532` to increase hard limit of stack size. Otherwise it will throw errors of `segmentation fault`.



```
admin@C02D36V0ML85 HWB % ulimit -s 65532
admin@C02D36V0ML85 HWB % gcc -DUSE_CLOCK seq_mm.c timing.c -o seq_mm
admin@C02D36V0ML85 HWB % ./seq_mm
****************************************************
||c||_F = 2630354827429312.50
Elapsed Time: 12.299 s.
admin@C02D36V0ML85 HWB % gcc -DUSE_CLOCK -O0 seq_mm.c timing.c -o seq_mm_O0
admin@C02D36V0ML85 HWB % ./seq_mm_O0
****************************************************
||c||_F = 2630354827429312.50
Elapsed Time: 11.926 s.
admin@C02D36V0ML85 HWB % gcc -DUSE_CLOCK -O3 seq_mm.c timing.c -o seq_mm_O3
admin@C02D36V0ML85 HWB % ./seq_mm_O3
****************************************************
||c||_F = 2630354827429312.50
Elapsed Time: 5.66807 s.
```



2.2. Optimization of Matrix Multiplication (15 points)

This exercise is intended to show how the reuse of data that has been loaded into cache by previous instructions can save time and thus increase the performance of your code.

`seq_mm.c` is a simple code that performs a 1,500 by 1,500 matrix multiplication. Develop a new version of the code that uses blocking to improve its temporal locality.

Use the following command to compile `seq_mm`

```
gcc -DUSE_CLOCK seq_mm.c timing.c -o seq_mm
```

Submission

- `P22.pdf`: Report with replicability information (see Submission note above), the improvements in elapsed execution time when using **separately** `-O0`, `-O3`, loop unrolling, blocking and unrolling/blocking. Please report your results in tabular form with columns corresponding to optimization flags or techniques as appropriate
- `P22.c`: Source code for the combined version of the code with loop unrolling and blocking. In your source code, please add comments to highlight where you have applied unrolling and blocking

Note: for "loop unrolling/blocking", unroll the blocking version OR unroll the innermost layer for x2 or x4.

Table for improvements in elapsed execution time (*unit: seconds*)

	-O0	-O3	4x loop unrolling	blocking	innermost layer x4 unrolling/blocking	No optimization, reference
seq_mm.c	11.926	5.66807	11.8876	9.47124	7.84174	12.299