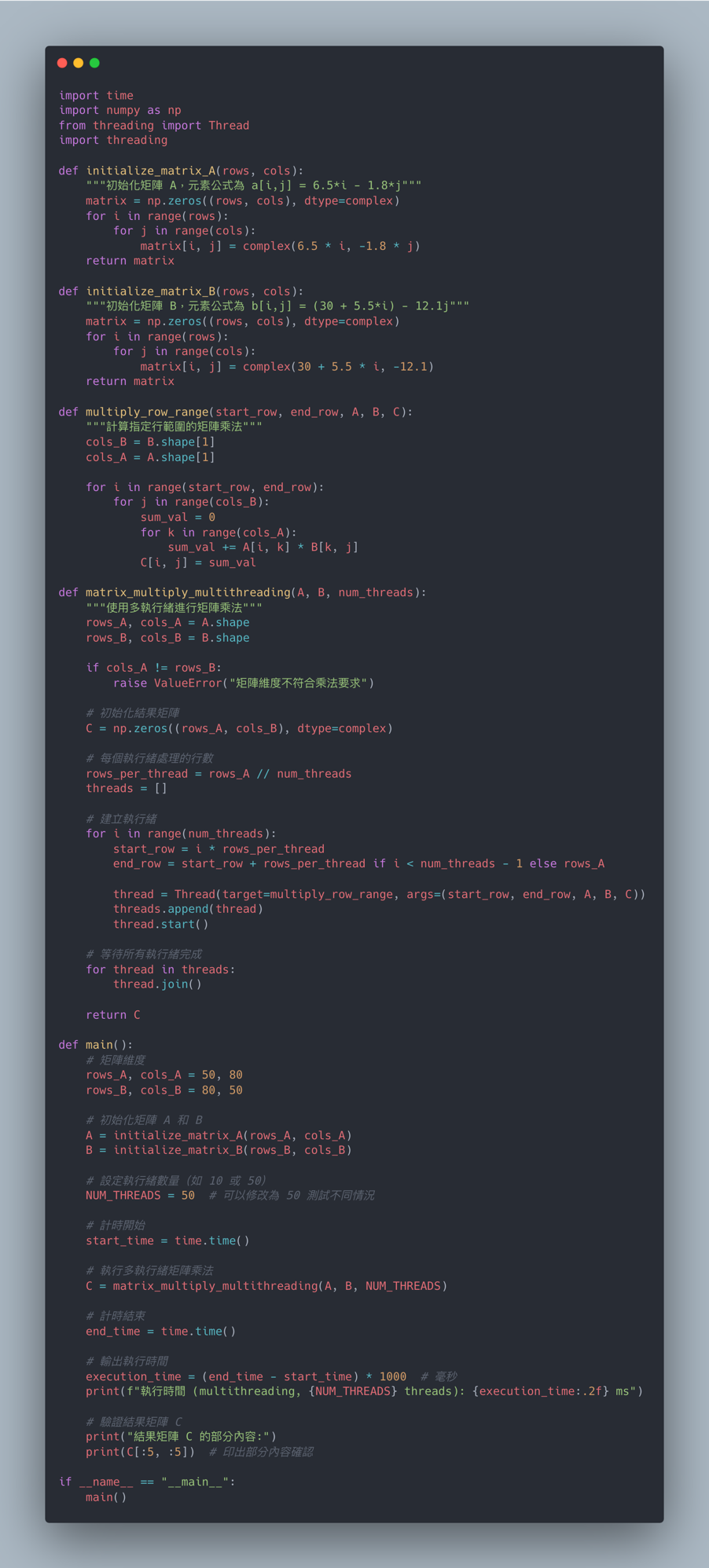
**National Taipei University of Technology OS - HW1**

**113368545 陳昀鴻 職電子碩一**

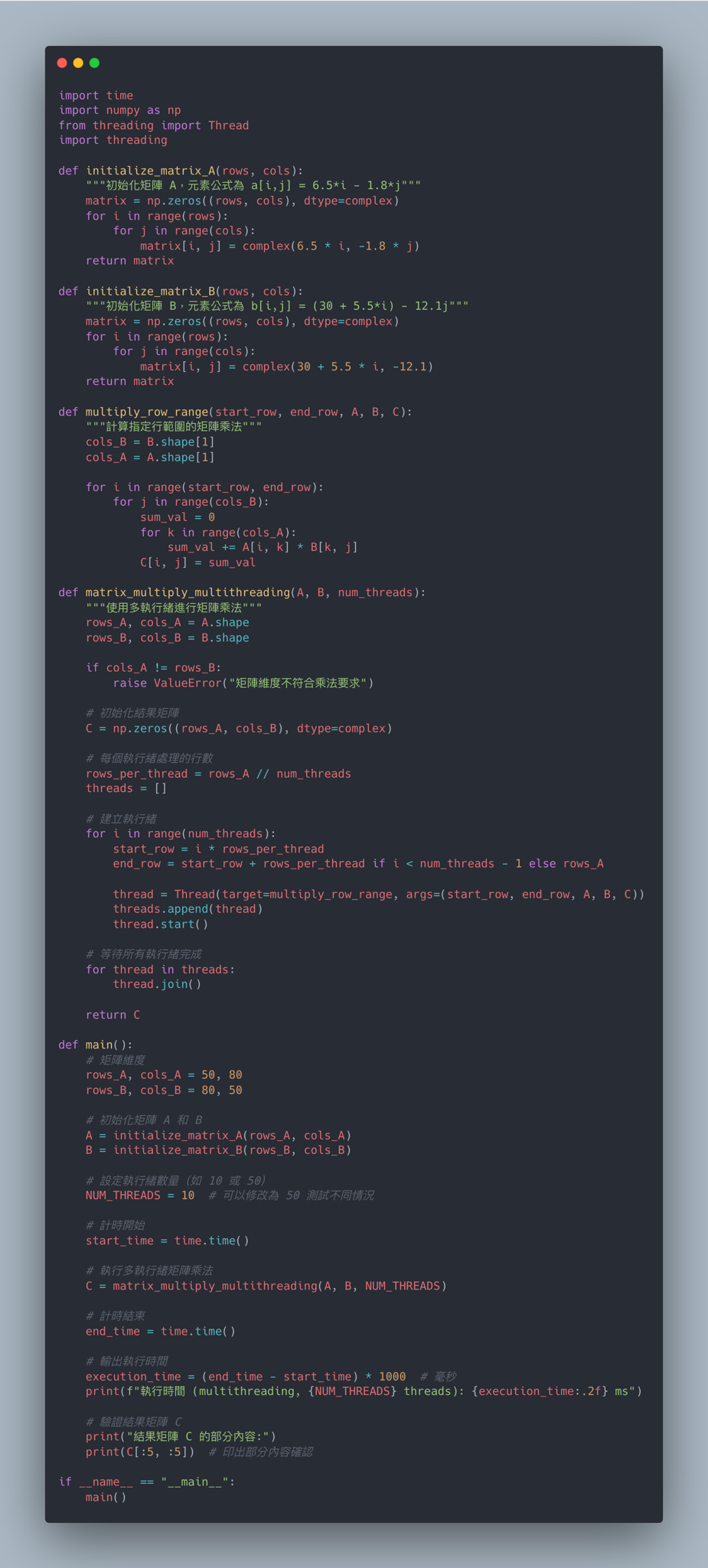
**Source Code**

1. 一張含有 文字, 螢幕擷取畫面 的圖片

   自動產生的描述**First Program – A (For-Loops)**
2. **Multithread( 50 threads)**



1. **Multithread( 10 threads)**



**Q1: Point out the *major parts* coded in the *threaded* program to highlight its differences with *for-loops*.**

**A1: 使用多執行緒的方式做矩陣乘法，並且是利用threading這個Library來實現。**

**一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述**

**其中，將其包裝成function，parameter帶入A、B矩陣，以及num\_threads 數量。**

**Q2: Record your experimental results at least 3 rounds execution in the below table, and state how you can count the running time of programs in ms.**

**A2: （比較表呈現）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Coding  Skill | No.  of  threads | Execution  Time  (ms) | | | Average Execution Time |
| For-Loops  【A】 | **1** | **26.70ms** | **29.08ms** | **28.41ms** | **28.06ms** |
| Multithread  【B1】 | **50** | **31.79ms** | **31.78ms** | **31.33ms** | **31.63ms** |
| Multithread  【B2】 | **10** | **30.08ms** | **30.23ms** | **30.22ms** | **30.17ms** |
| Differences  【B1-A】 | **49** | **5.09ms** | **2.7ms** | **2.92ms** | **3.57ms** |
| Differences  【B2-A】 | **9** | **3.38ms** | **1.15ms** | **1.81ms** | **2.11ms** |

用Python的time來計算運算時間，計算具體步驟從在執行矩陣乘法之前就開始計時，最終計算完成之後計算結束時間。

Step : 矩陣乘法前 -> （執行矩陣乘法） -> 結束矩陣乘法運算

**Q3: State your discovering and comments on this exercise of coding threaded programs.**

**A3: 由A2比較表可發現差異性並不大。**

**這個問題有幾個主要原因：**

1. **矩陣太小**

目前的矩陣大小是 50x80 和 80x50，對於這麼小的矩陣，多處理程序的開銷反而大於計算時間，處理程序的創建和管理需要額外的時間成本。

1. **不應該使用Multithread 而應該使用 Multiprocess**

因為矩陣運算應該是CPU處理而不是I/O處理。

對此如果我們將矩陣放大，來驗證看看是否有把計算時間下降，如以下比較表：

**以500x500的矩陣我們可以獲得以下的比較表：(Multithread)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Coding  Skill | No.  of  threads | Execution  Time  (ms) | | | Average Execution Time |
| For-Loops  【A】 | **1** | **17616.47ms** | **17444.21ms** | **17559.33ms** | **17540.00ms** |
| Multithread  【B1】 | **50** | **16975.51ms** | **17434.56ms** | **17433.14ms** | **17281.07ms** |
| Multithread  【B2】 | **10** | **18014.88ms** | **17774.77ms** | **17553.43ms** | **17781.02ms** |
| Differences  【B1-A】 | **49** | **640.96ms** | **9.65ms** | **126.19ms** | **258.93ms** |
| Differences  【B2-A】 | **9** | **398.41ms** | **330.56ms** | **5.9ms** | **244.95ms** |

我們可以發現，具體其實並沒有非常顯著的改善執行時間。透過上述比較圖，我們可以發現，不一定使用Multithread來優化會是最佳策略。

其中Python當中的Multithread應該應用在I/O密集處理的操作（如：檔案操作、網路請求），所以用Multiprocessing比較合理，因為：CPU密集處理（如：矩陣運算、大型數據處理），最後可以透過share memory \ pipeline \ IPC 等方式來通信。

* **GIL 是什麼？**

GIL 是 Python 的全域直譯器鎖（Global Interpreter Lock），它是 CPython（Python 的標準實現）中的一個互斥鎖，用於限制同一時間只能有一個執行緒執行 Python 位元組碼。

**重點程式碼如以下：**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述**

多執行緒當中我做了一個C\_shared memory 來做一個共享記憶體空間，並且分配每個process應該執行的行數，然後最終將結果回傳回來的時候轉換成numpy矩陣。

而矩陣計算的地方我寫成一個function作為包裝，parameter有三：A矩陣、B矩陣、process 的個數（threads -> process）。

**以500x500的矩陣我們可以獲得以下的比較表：（Multiprocessing）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Coding  Skill | No.  of  threads | Execution  Time  (ms) | | | Average Execution Time |
| For-Loops  【A】 | **1** | **17415.47ms** | **17523.90ms** | **17359.17ms** | **17532.84ms** |
| Multiprocessing【B1】 | **50** | **4494.03ms** | **4096.50ms** | **4233.13ms** | **4274.55ms** |
| Multiprocessing【B2】 | **10** | **3859.10ms** | **3854.23ms** | **3807.99ms** | **3840.44ms** |
| Differences  【B1-A】 | **49** | **12921.44ms** | **13427.4ms** | **13126.04ms** | **13154.96ms** |
| Differences  【B2-A】 | **9** | **13556.37ms** | **13669.67ms** | **13551.18ms** | **13592.40ms** |

**Running Screenshot**

**一張含有 文字, 軟體, 多媒體軟體, 網站 的圖片

自動產生的描述**

**一張含有 文字, 軟體, 多媒體軟體, 網站 的圖片

自動產生的描述**

**環境**

Hardware:

Hardware Overview:

Model Name: MacBook Pro

Model Identifier: Mac15,3

Model Number: Z1C80002RTA/A

Chip: Apple M3

Total Number of Cores: 8 (4 performance and 4 efficiency)

Memory: 16 GB

System Firmware Version: 10151.121.1

OS Loader Version: 10151.121.1

Serial Number (system): CJ492309HN

Hardware UUID: 3650AB42-B261-5140-93F0-B5A49288DA1E

Provisioning UDID: 00008122-000571C11A40001C

Activation Lock Status: Enabled

**結論**

在看到題目的同時，第一直覺會想用Thread去解決，但轉念一想他所做的事情應該是針對CPU處理的部分（矩陣運算）而並非I/O運算，而做完比較表之後，也實際採取將Multithread的架構改成用Multiprocessing的方式來做處理。

最終也確實證明在本案例當中，使用Multiprocessing會更為高效的表現。

**Github Link :** [**https://github.com/roger28200901/NTUT\_OS**](https://github.com/roger28200901/NTUT_OS) **（source treee & 報告原檔）**