作業系統 HW1

數據所 310554022 張凱雋

A. 實驗環境

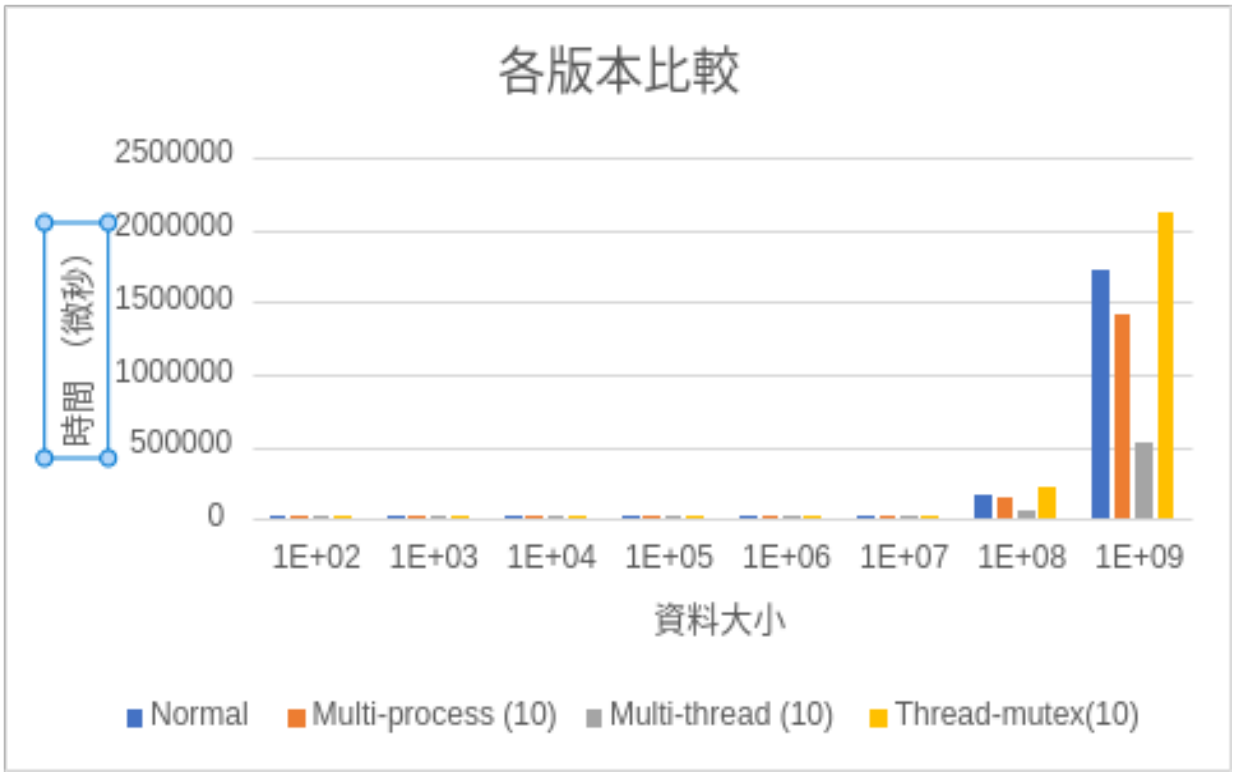
- CPU Core : 4 ; Total threads : 8

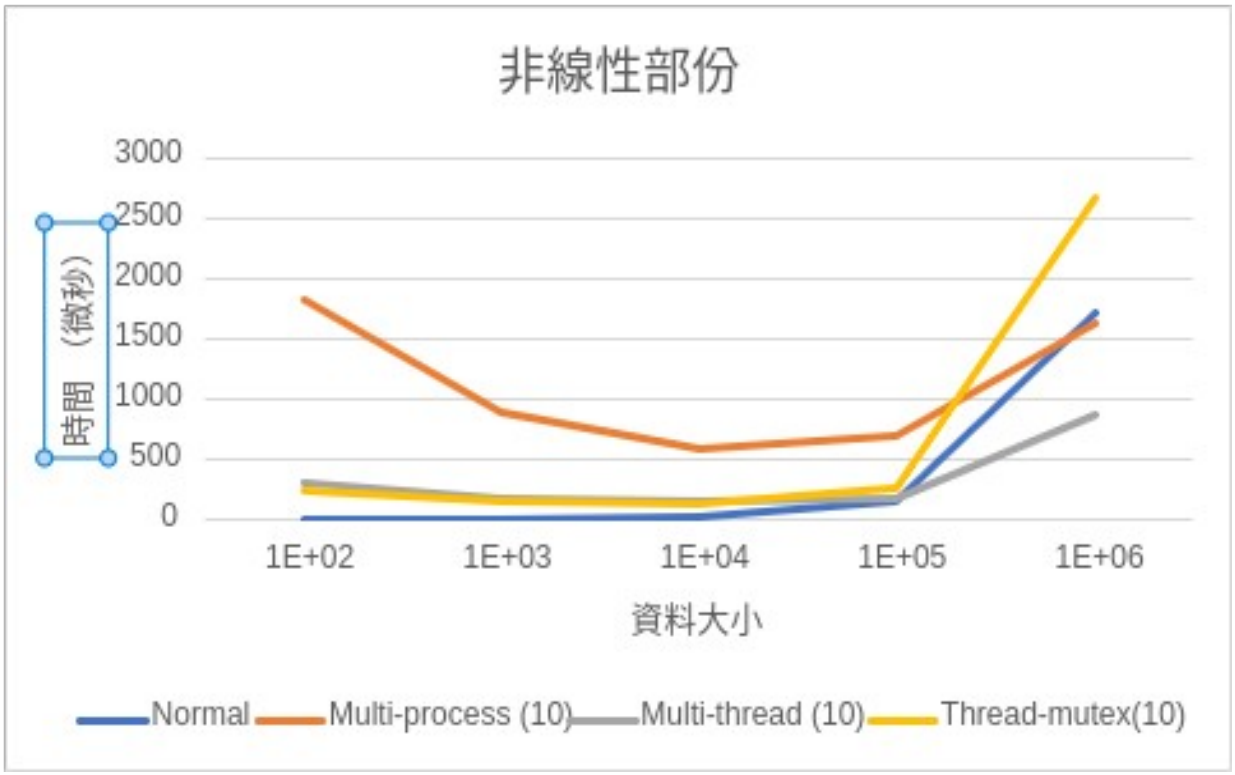
B. 各版本之間的比較

- 陣列資料大小: 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000
- 測試版本：Normal, Multi-process (10), Multi-thread (10), Multi-thread-mutex (10)
- 實驗方法：每個實驗皆會測試 10 次，最後的結果為這 10 次測驗的平均

實驗結果:

	Normal	Multi-process (10)	Multi-thread (10)	Thread-mutex (10)
1E+02	1.00	1835.20	312.00	238.90
1E+03	3.80	883.90	176.60	161.70
1E+04	19.20	600.40	151.60	144.20
1E+05	168.20	702.10	184.10	272.00
1E+06	1714.10	1634.10	862.40	2678.60
1E+07	17486.20	12652.30	6197.90	22164.40
1E+08	172636.10	155571.40	56896.40	213406.20
1E+09	1720471.50	1417259.60	523556.80	2114747.90





實驗結果分析:

1. 資料大小的影響

在資料數很少的時候（例：100，1000），Normal 版本所花費的時間反而是最少的，可能是因為要產生一個 child process 或一個 thread 所需要的時間成本大於原本的計算成本，所以在時間方面原本 sequential 的方法會較好。

在經過實驗之後，當資料數量超過 1000000 後，Multi-process 與 Multi-thread 的效能才會漸漸超過 Normal 的方法，並且 Multi-thread 的表現會最好。但是也有可能是我 child process 數目設定的不好的關係，於是我在下一個部份會做不同 child process 數目之間的比較。

2. Thread 用 mutex 後的影響

因為在資料量較少的時候，產生的成本可能佔了較大的影響，所以我以大資料量為主去分析。在資料量大的時候，使用 mutex 後所需要的時間遠遠大於原本的 multi-thread，甚至也超過了 normal 所花費的時間，可見使用 mutex 做 lock 後會造成效能的下降。

Mutex 的正確性驗證:

```
Array size: 10000000
Test 0: time spent: 16995 usec | Found "0" 11 times.
Test 1: time spent: 17521 usec | Found "0" 11 times.
Test 2: time spent: 17119 usec | Found "0" 11 times.
Test 3: time spent: 17253 usec | Found "0" 11 times.
Test 4: time spent: 17568 usec | Found "0" 11 times.
Test 5: time spent: 16926 usec | Found "0" 11 times.
Test 6: time spent: 17348 usec | Found "0" 11 times.
Test 7: time spent: 17077 usec | Found "0" 11 times.
Test 8: time spent: 17338 usec | Found "0" 11 times.
Test 9: time spent: 17062 usec | Found "0" 11 times.
Average time of Normal:            17220.70
```

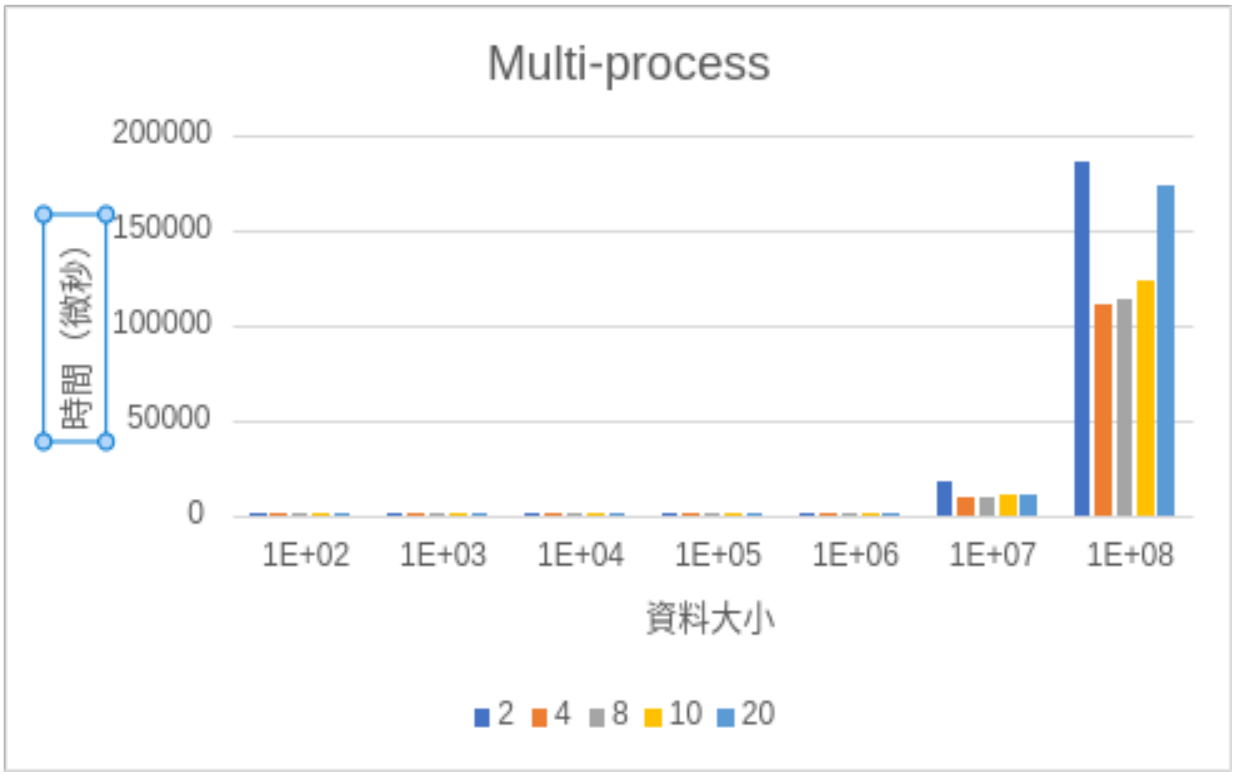
```
Test 0: time spent: 21891 usec | Found "0" 11 times.
Test 1: time spent: 23041 usec | Found "0" 11 times.
Test 2: time spent: 21775 usec | Found "0" 11 times.
Test 3: time spent: 22100 usec | Found "0" 11 times.
Test 4: time spent: 22698 usec | Found "0" 11 times.
Test 5: time spent: 22008 usec | Found "0" 11 times.
Test 6: time spent: 22123 usec | Found "0" 11 times.
Test 7: time spent: 22457 usec | Found "0" 11 times.
Test 8: time spent: 22026 usec | Found "0" 11 times.
Test 9: time spent: 22391 usec | Found "0" 11 times.
Average time of Multi-Thread-Mutex: 22251.00
```

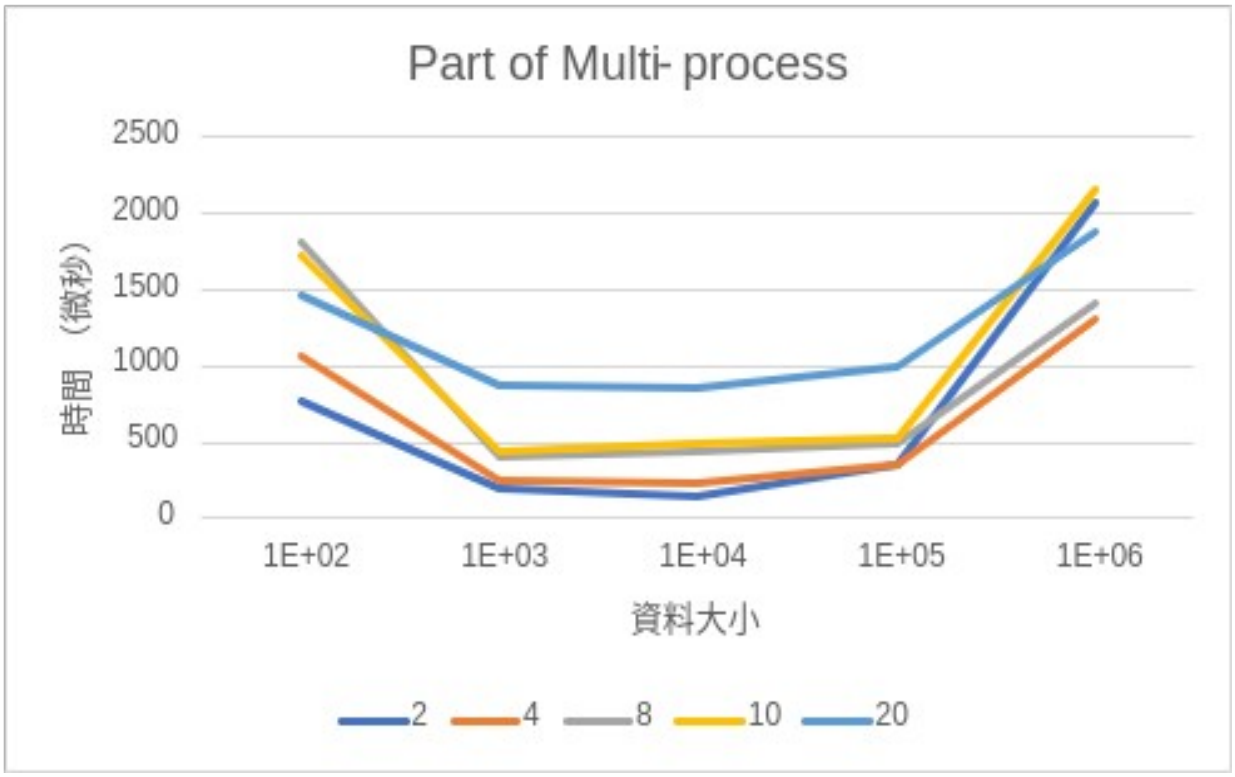
C. Multi-process child 數目之間的比較

- 陣列資料大小: 100, 1000, 10000, 100000, 1000000, 10000000, 100000000
- 測試 child 數目：2, 4, 8, 10, 20
- 實驗方法：每個實驗皆會測試 10 次，最後的結果為這 10 次測驗的平均

實驗結果:

	2	4	8	10	20
1E+02	772.7	1058.6	1818.2	1726.1	1472.3
1E+03	191	254.4	406.9	435.5	872.7
1E+04	145.1	234.4	432.7	492.5	861.4
1E+05	356.9	360.9	488.8	519.2	1003.8
1E+06	2066.1	1303.6	1419.6	2164.9	1874.6
1E+07	18509.1	10184.5	10625	11200.6	11717
1E+08	186261.7	111094.5	113929.3	123800.3	174380.7





實驗結果分析:

1. Multi-process child 數目的影響

經過實驗後發現普遍在 process 數量為 4 時的表現最好，而因為我的實驗環境核心數只有 4，所以這個結果也算合理。 若是 process 的數量太少，可以參照上一部份的 normal，在資料量大時的花費時間會遠大於其他。

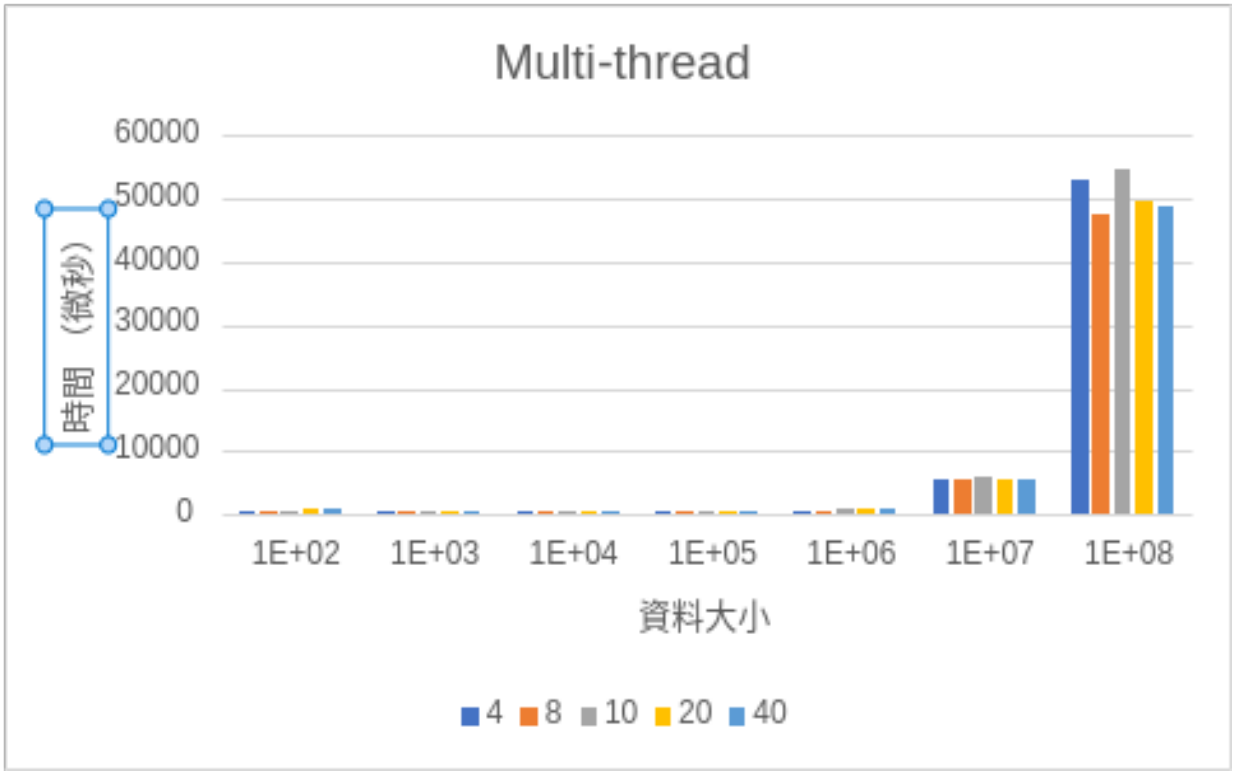
若是 process 數目太大也會對結果造成影響。當 process 數為 20 時，資料量很大時會大大降低其效率，使得花費的時間大大上升。此結果可能是因為要核心數太少需要頻繁的在 processes 之間做 context switch 的關係，使得花費的時間大幅上升。

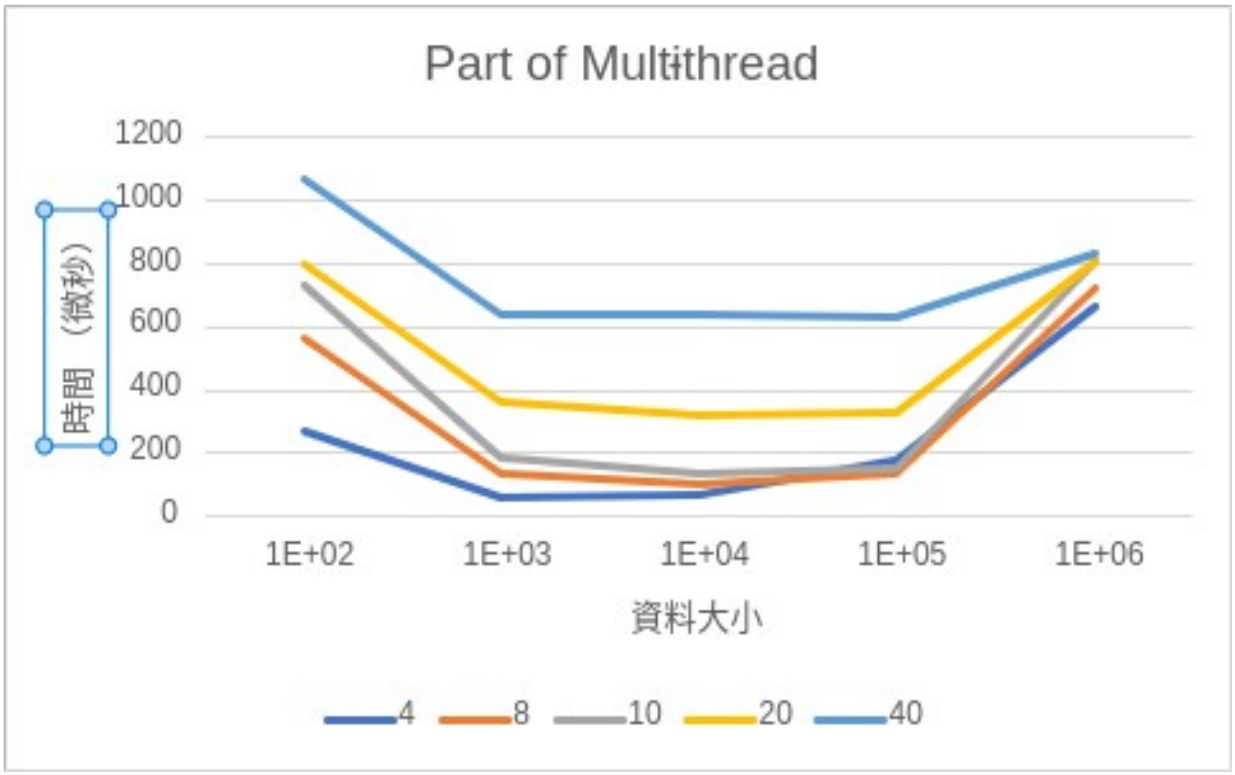
D. Multi-thread Number 之間的比較

- 陣列資料大小: 100, 1000, 10000, 100000, 1000000, 10000000, 100000000
- 測試 thread 數目：4, 8, 10, 20, 40
- 實驗方法：每個實驗皆會測試 10 次，最後的結果為這 10 次測驗的平均

實驗結果:

	4	8	10	20	40
1E+02	271	567.1	735.1	804	1072
1E+03	62.5	139.7	183.8	362.5	644.1
1E+04	65.6	102.2	136.9	317.5	638.3
1E+05	176.4	135.3	155.5	326.1	633.3
1E+06	666.4	723.2	810.3	808.9	833.2
1E+07	5647.3	5367.3	6155.6	5711.1	5436.5
1E+08	53044.7	47740.9	54728.2	49540.6	49050





實驗結果分析:

1. Multi-thread 數目的影響

感覺差不多，影響不是很大。當 thread 數目為 8 時，表現最好，此結果符合了實驗環境的設定。但從結果上來看，盲目的增加 thread 的數量並不會使其效能呈倍數成長。