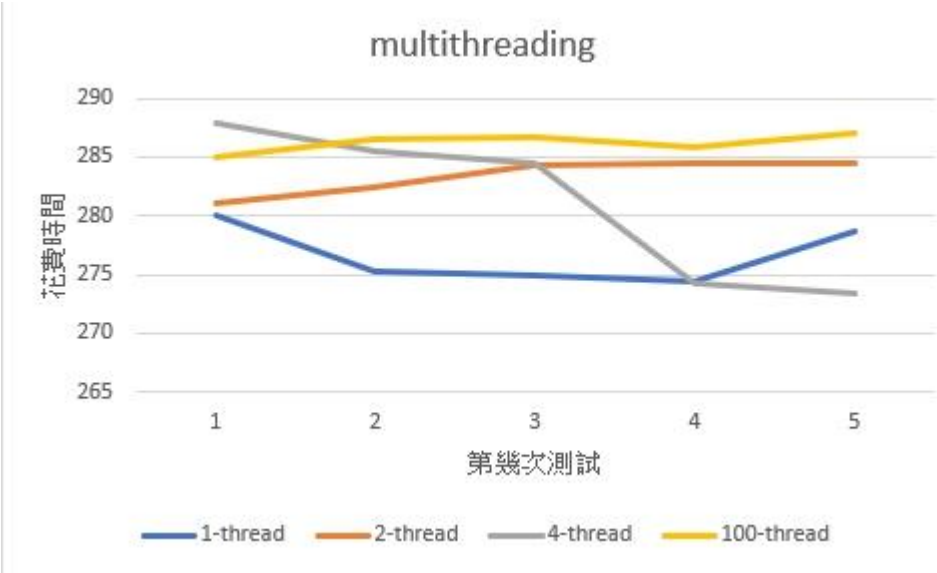
使用 python3.8.2

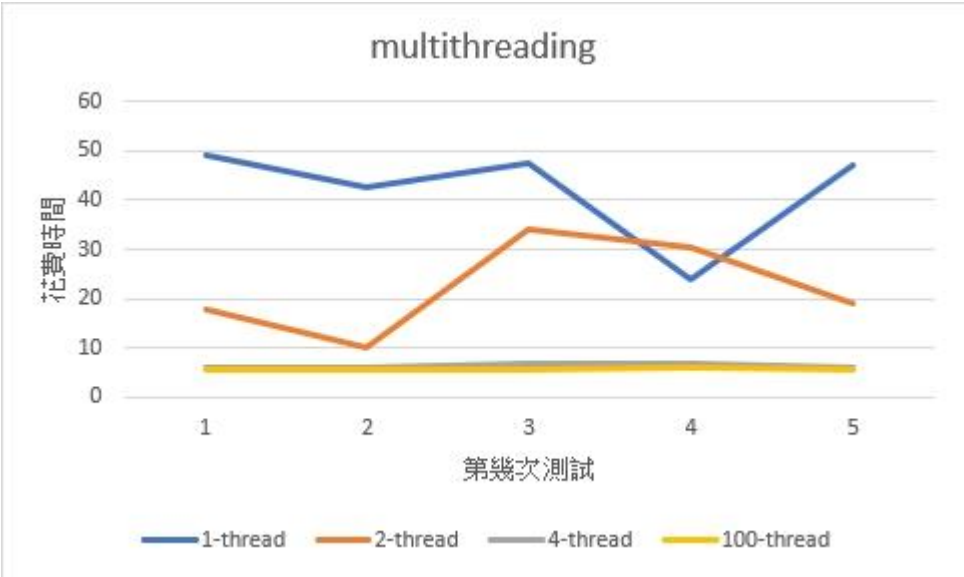
一、執行緒數量對效能的影響

Task1:



thread	1-thread	2-thread	4-thread	100-thread
1-st test	279.961	281.0506	287.8279	285.0152
2-nd test	275.267	282.4544	285.4681	286.5898
3-rd test	274.949	284.2421	284.514	286.6629
4-th test	274.353	284.53	274.276	285.8186
5-th test	278.597	284.53	273.3687	287.0804
average	276.6254	283.36142	281.09094	286.23338

Task2:

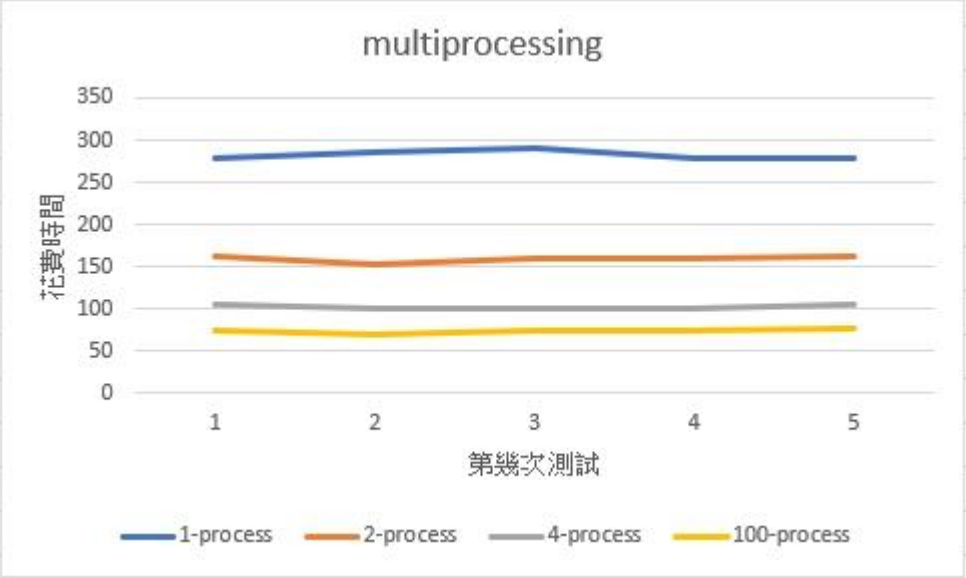


thread	1-thread	2-thread	4-thread	100-thread
1-st test	49.1737	17.6871	5.9701	5.6732
2-nd test	42.598	10.2638	6.1276	5.6568
3-rd test	47.5406	34.1834	6.9612	5.4836
4-th test	23.9961	30.5873	6.8084	5.9773
5-th test	46.8904	19.0112	5.9761	5.7316
average	42.03976	22.34656	6.36868	5.7045

task1 跟 task2 的結果有蠻大的差異，在 task1 中反而花費時間最低的是 thread 為 1 的時候，花費時間最高的則是 thread 數為 100 的時候；在 task2 中花費時間最低的則是 thread 為 100 的時候，花費最高的則是 thread 數為 1 的時候。而我的推測是在 task1 中會使用相同部分的資源較多，並且執行單次工作時所需的資源也多，因此會發生資源搶奪的現象，也因此當 thread 數高時反而會增加執行所需的時間，而 task2 中各自 thread 使用的資源不重複也較少，也因此當 thread 數提高時便會提高執行的速度。

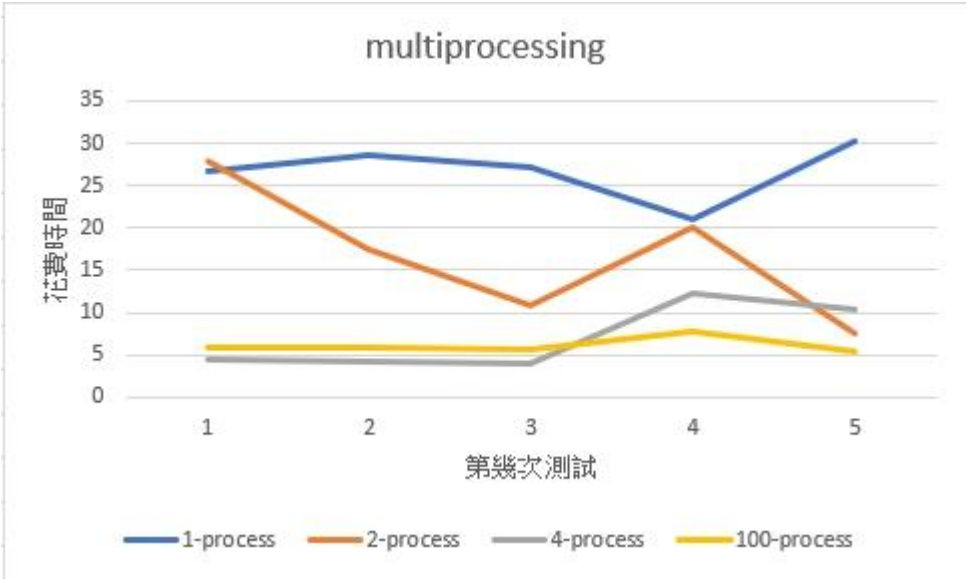
二、行程數對效能的影響

Task1:



process	1-process	2-process	4-process	100-process
1-st test	278.209	161.2535	106.0549	74.8409
2-nd test	286.2545	153.111	99.488	69.3449
3-rd test	291.4399	160.4572	100.151	75.2732
4-th test	278.7637	159.7352	100.7332	75.0109
5-th test	278.5345	161.2083	105.2648	76.9429
average	282.64032	159.15304	102.33838	74.28256

Task2:

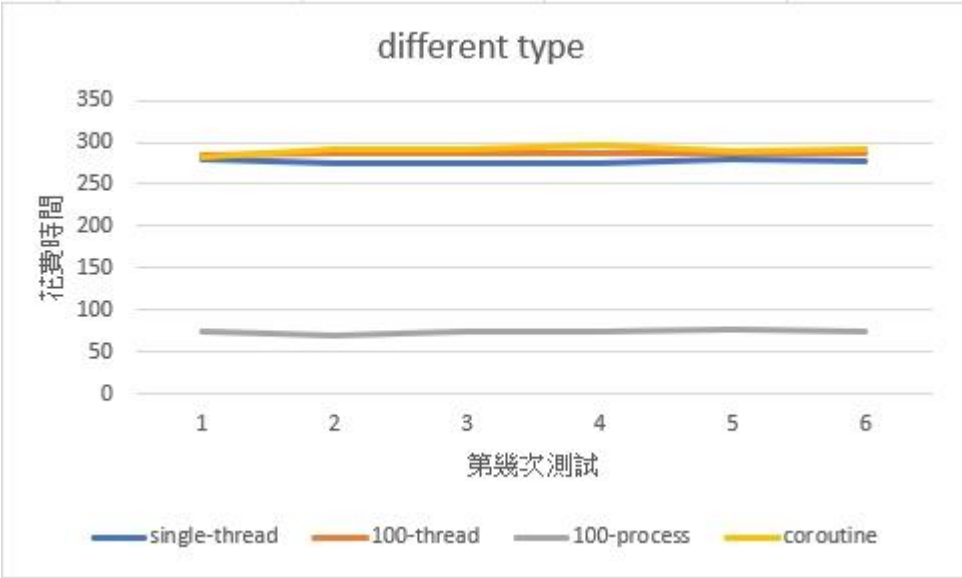


process	1-process	2-process	4-process	100-process
1-st test	26.7531	27.8291	4.3162	5.8865
2-nd test	28.6599	17.5267	4.0825	5.8099
3-rd test	27.0937	10.8022	4.0746	5.5384
4-th test	21.0329	20.0108	12.2578	7.7733
5-th test	30.2237	7.5602	10.3976	5.3822
average	26.75266	16.7458	7.02574	6.07806

從 task1 的角度來看，很明顯的當 process 為 100 時會最快，而單一 process 時最慢，task2 來看則是 process 數是 4 與 100 並無明顯差異，也因此可以推斷 task1 的平均執行時間皆較常而 task2 較短，所以當執行時間小到一個定值的時候便會有邊際效應而不會繼續減小執行時間。

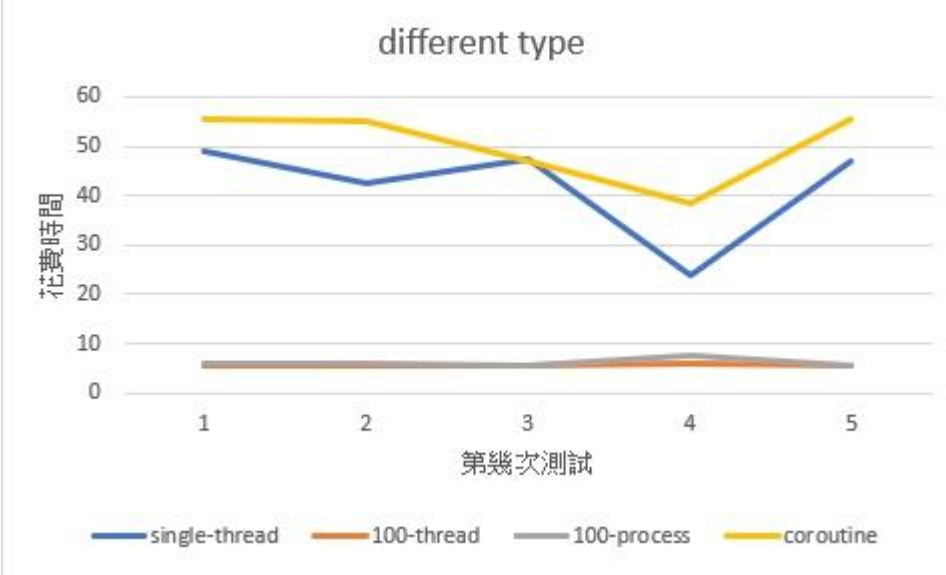
三、多執行緒、多行程、協程的效能比較

Task1:



compare	single-thread	100-thread	100-process	coroutine
1-st test	279.961	285.0152	74.8409	283.1399
2-nd test	275.267	286.5898	69.3449	292.2447
3-rd test	274.949	286.6629	75.2732	292.7846
4-th test	274.353	285.8186	75.0109	295.6871
5-th test	278.597	287.0804	76.9429	289.0913
average	276.6254	286.23338	74.28256	290.58952

Task2:



compare	single-thread	100-thread	100-process	coroutine
1-st test	49.1737	5.6732	5.8865	55.3281
2-nd test	42.598	5.6568	5.8099	54.9763
3-rd test	47.5406	5.4836	5.5384	46.9664
4-th test	23.9961	5.9773	7.7733	38.5622
5-th test	46.8904	5.7316	5.3822	55.328
average	42.03976	5.7045	6.07806	50.2322

在 task1 中，花費時間最低的是 100 processes 的時候，其餘三者的花費時間則皆差不多，在 task2 中，花費時間最低的則是 100 processes 與 100 threads 的時候，花費最高的時候則會是 coroutine 的狀況，也因此可以推斷 100 processes 執行各個 task 的執行效率最高，100 threads 則要根據該 task 的狀況來判定其所花的時間為長或是短，而 coroutine 則是在各個 task 中執行效率最慢的；我的推斷是因為 coroutine 的執行方式為暫停並將執行權讓給其他 Coroutine，然後等其執行完後再繼續執行，因此在 idle time 較短的 task 中會較慢，因為會需要許多不必要的等待時間與 coroutine 之間的切換時間，但若在 I/O 等需要 idle 的 task 中則有可能會較其他執行方式效率高。