Parallel TSP

107062338 邱俊維

Motivation、

在找尋適合做為 Final project 的題材時看到了 TSP，在過往學習演算法時便有學習到 TSP 的 optimal version 為 NP-hard 的問題。而目前對於 NP-hard 問題並沒有 polynomial time 的演算法，因此勢必在 input size 大時會需要大量的時間。若能將計算平行處理的話，可以節省許多時間，因此決定以 TSP 作為核心來進行平行。

What is TSP、

Travelling salesman problem(TSP)，給定數個城市兩兩之間的距離，求出從一起點城市出發，拜訪過所以其他城市”一次”後回到起點城市。TSP 可以使用 graph 來表示，G = (V, E)，V 為城市所成之集合，兩城市 a, b ∈ V 之間的距離則形成一條 edge e_{ab} ∈ E, w(e_{ab}) = a, b 城市距離。

2-opt Algorithm、

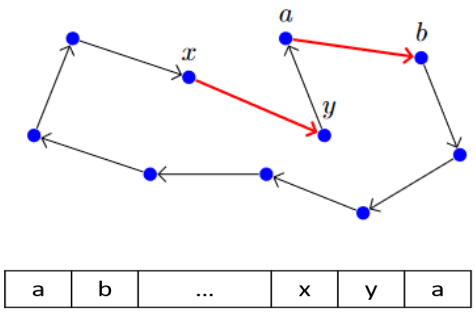
2-opt 於 1958 年由 G. A. Croes 提出（G. A. Croes, A method for solving traveling salesman problems. Operations Res. 6 (1958) , pp., 791-812.）。其主要概念為隨機挑選 2 點，藉由挑選中的點來進行優化。

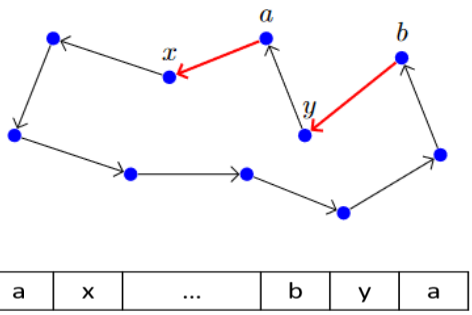
Find the sequence $s_0 \in S$ for which $M(s_0) = \min_s M(s)$, where the measure $M(s)$ of the sequence s is defined by $M(s) = \sum_{(i,j)\in s} c_{ij}$.

1. Find a trial solution $s \in S$, for which $M(s)$ is as small as we can make it at a first try.

2. Apply simple transformations, called ‘inversions,’ which transform this trial solution into other elements of S, whose measures are progressively smaller.

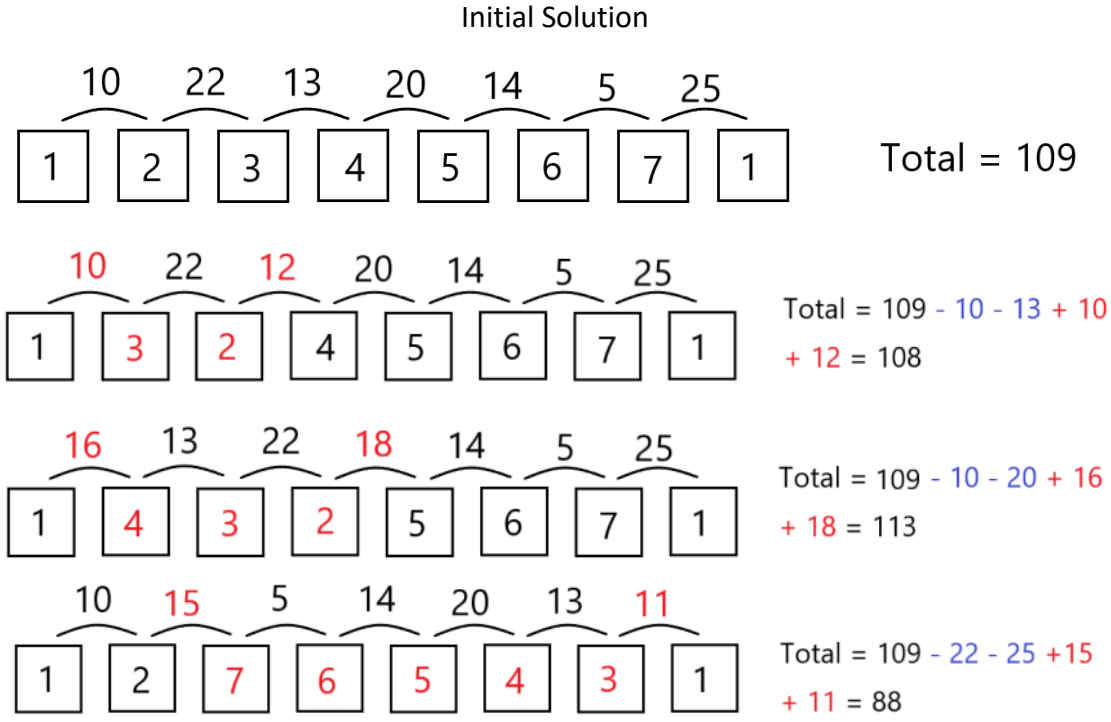
根據 2-opt 的概念，我們先取出一個隨機的 solution，使用 1D-array 來表示該 solution，假設該 array 為 a, b, c, d, e, f, g, a，取非 start/end 的隨機兩個點如 c, g，將其進行反轉得到 new solution array：a, b, g, f, e, d, c, a，計算 new solution 的 cost 是否優於原本的 solution。若較優，則 new solution 取代原本的 solution；反之則找尋其他隨機兩點進行嘗試。重複上述步驟直到找不到更佳的 solution。





以上圖為例，左圖的路徑為 a→b→…→x→y→a，我們選取該路徑上的兩點 b, x 作為進行優化的點。則我們會將路徑中的 b→…→x 反轉，得到 x→…→b，因此我們會得到新的路徑 a→b→…→x→y→a，計算新路徑的 distance 並與原先的 distance 做比較，若得到較好的結果則採用新的結果。

2-opt Example、



在上述例子中，選定兩點 i, j 來進行計算新的 Total distance，計算時減去的數字為選定的兩點為 i 與 i - 1 的實際距離及 j 與 j + 1 的實際距離(藍字)，加上的數字為 i 與 j + 1 的實際距離及 j 與 i – 1 的實際距離。嘗試所有可能後，找到最佳的新 distance，並且將 array 進行對應的 reverse。

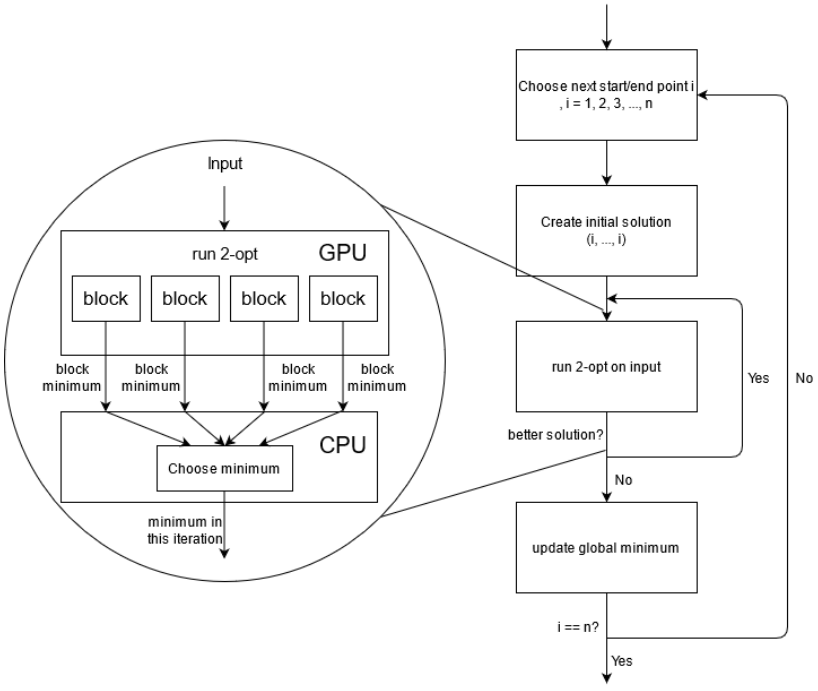
Advantage、

使用 2-opt 的好處是找尋 new solution 時僅需簡單的運算即可得到 new solution 的 cost。並且在找到當前 solution 經 inversion 後可達的最佳答案前，皆不須對當前的儲存 solution 的 array 做實際的搬移，而是當求出最佳的 new solution 後再根據該 solution 所記錄的 index 值來對 array 做 inversion。

2-opt pseudo code、

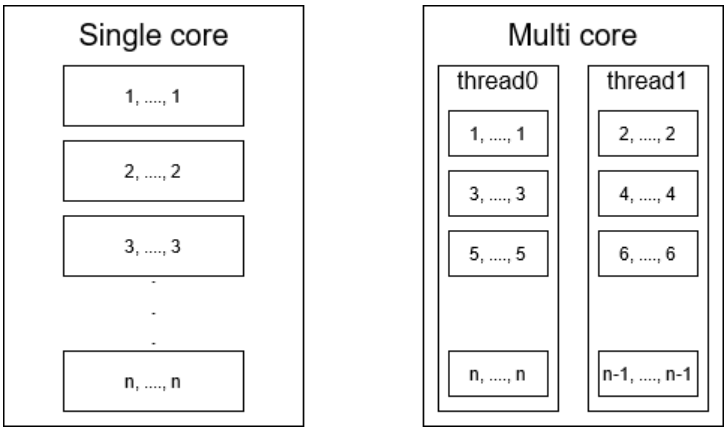
```
tour = initial solution;
index_i, index_j;
min_dist = infinite;
for(i = 1 to n - 3){
    for(j = i + 1 to n - 2){
        l_dist = dist[tour[i - 1]][tour[j]] + dist[tour[i]][tour[j + 1]]
               - dist[tour[i - 1]][tour[i]] - dist[tour[j]][tour[j + 1]])
        if(l_dist < min_dist){
            min_dist = l_dist;
            index_i = i, index_j = j;
        }
    }
}
reverse(tour, index_i, index_j);
```

2-opt based TSP algorithm flow chart、



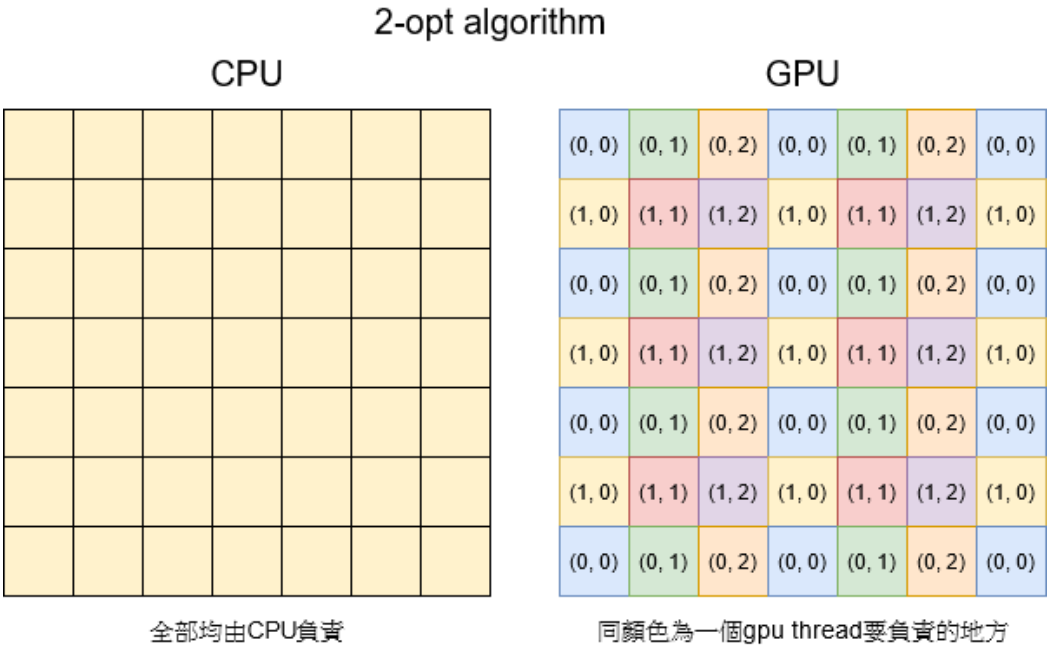
Parallel、

上圖的演算法中共分成兩部分進行平行，第一部分為 choose next start/end point 及 create initial solution，假設總共有 n 個 cities，則總共需計算 start/end point i, i = 1, 2, 3, …, n，共 n 種。將此 n 種分給 pthread 去做，則每個 pthread 需負責計算 n / num_threads 個 start/end points。



每一個 pthread 使用一個 GPU，pthread 在 Run 2-opt 的部分會把 function assign 給 GPU 來進行加速。每個 gpu thread 會負責計算數個 i, j 的組合，每個 gpu thread 會得到嘗試過的 i, j 組合中的最佳解 thread minimum，而每個 gpu block 會比較 block 內所有 thread minimum，最後得到 block minimum。gpu 完成上述運算後，將所有的 block minimums 搬回 Host 端，最後交由 CPU 去挑選出該次 iteration 中最佳的 new solution。

Iteration 內得到的 new solution 若優於前次，則採用 new solution 進入下一次的 iteration，直到執行 iteration 後無法獲得更好的 solution 時便結束。(也可 static 設定 iteration 執行的次數，Ex: iteration 執行 10 次，此作法 scalability 較好)



最後會進到 update global minimum 的階段，比較以 i 作為 start/end point 的結果是否有比其他 start/end point 的 solution 更佳，若有則更新 global minimum。

Experimental results、

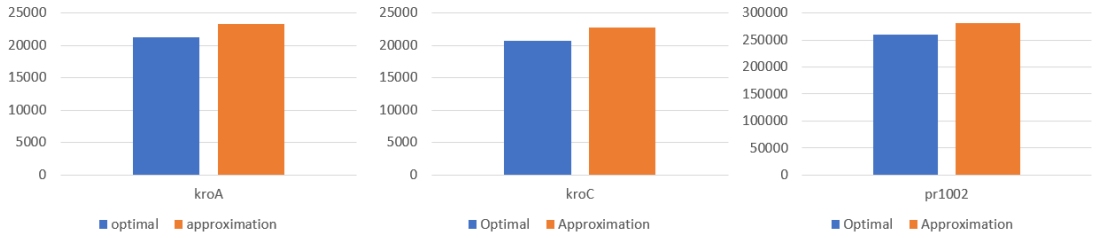
testcase	size	Sequential(microseconds)	Parallel(micro seconds)	Speedup
A16	16	102	179298	0.0005
A48	48	17202	187790	0.0916
kroA100	100	21285	294358	0.0723
kroC100	100	20750	279441	0.0742
Gr202	202	674327	322955	2.0879
Pcb442	442	18585208	1789195	10.3874
Uy734	734	345170616	19866691	17.3743
Dsj1000	1000	2962227815	117690563	25.1696
Pr1002	1002	323010587	8185040	39.4635
U1817	1817	4561166105	80848123	56.4164

Execution time

在上圖中可以看到當 input size 很小時，sequential 的速度快於 parallel program，因 parallel program 需要進行 pthread 的 create 以及 pthread 裡會進行搬移資料至 device 端以及 assign task 給 gpu，整個程式的執行時間有一大部分都在執行上述的操作，真正計算的時間相對少了許多。

但在 input size 越來越大時，使用 pthread 及 gpu 進行加速的效果便開始顯現出來。以 u1817 為例，sequential program 執行時間多達 4561 秒(約 1 小時 16 分鐘)，而在 parallel program 只需要 81 秒(1 分 21 秒)，兩者差距極大。實際上還有許多 size 更大的 testcases，但由於在 hades 上執行時間限制為 15 分鐘，因此就沒有對更大的 testcases 進行紀錄。不過曾嘗試執行 1900 多 cities 的 testcase，拆分成兩次執行的話在 parallel program 上總共約需要 20 分鐘，足見 TSP 在 input size 增加時，time complexity 的增加相當明顯。

testcase	Optimal	Approximation	Approximation/Optimal
kroA100	21282	23294	1.0945
kroC100	20749	22773	1.0975
pr1002	259045	280797	1.0839



Approximate ratio

在執行 approximation algorithm 時，通常會關心該 algorithm 能得到多好的答案，因此找了幾個有 optimal solution 的 testcases 來進行觀察，在上面所呈現的 cases 中有還不錯的結果。

testcase	size	Pthread0(microseconds)	Pthread1(microseconds)	pthread0/ pthread1
A16	16	234235	236377	0.9909
A48	48	23479	26074	0.9004
kroA100	100	118460	115814	1.0228
kroC100	100	116728	115968	1.0065
Gr202	202	182703	184574	0.9898
Pcb442	442	1664185	1639066	1.0153
Uy734	734	19377883	19338121	1.0020
Dsj1000	1000	116759269	116515166	1.0020
Pr1002	1002	8343575	8315826	1.0033
U1817	1817	81333313	80597552	1.0091

Loading balance

Pthread 雖然是進行 static assign，表面上看起來可以達到很好的 balance，但須注意的是在重複執行 2-opt 的時候，結束的條件是當執行完 2-opt 後沒有找到更好的答案才結束。因此實際上不同的 start/end point 所要執行 2-opt 的次數並不固定（若使用上述所說的直接設定 iteration 的次數而非執行到沒有更佳解才結束的話則沒有此問題）。不過觀察上表可以發現儘管每個 start/end point 所需要時間不相同，但在加總 thread 負責的所有 start/end points 花的時間後，兩個 thread 的花費時間不多。

Issue、

目前關於 project 有三個可以討論的部分。

一、在計算由 i 為 start/end point 的最佳答案過程中會重複不斷的 assign 2-opt task 給 gpu，大量的在 host 與 device 端的來回會花蠻多的時間在轉換上，因此會減少 performance。

在 presentation 時教授有建議若將整個 flow 搬至 gpu 上執行是否可以避免掉上述的情況，獲得更好的執行時間。個人的看法是此方法是有機會提升 performance，但因目前是將一些較瑣碎的部分交由 host 處理，device 端專心執行 2-opt，若將瑣碎的部分也全部移到 device 端的話，device 端的執行效率也會受到影響，因此 performance 能否提升以及能提升多少可能還需要經過實驗後才有較確切的答案。

二、此次 project 將 TSP 當作所有兩兩城市間皆有路徑可到達，即 TSP 所成的 graph 為 complete graph。當 input size 增大時，會需要大量的記憶體來存放 distance 的資料，將 distance 搬至 GPU 的 global memory 後，因 shared memory 沒辦法一次容納如此大量的資料，會增加計算時需要向 global memory 存取的次數，進而導致執行時間上升。

三、在 input size 較小時 gpu 的 utilization 偏低，以我實際程式裡的設計，block size 為 16*16，若此時餵進的 input file 為 a16(16 個 cities)，會連一個 block 都佔不滿，更遠小於 gpu 同時可執行的 threads 數量。
針對此項問題教授提議若使用 asynchronous kernel 是否能夠增加 utilization。經過思考後認為的確對於增加 utilization 蠻有幫助，一開始的想法是在跑多次 2-opt 的部分進行 streaming，但後來注意到在執行數個 2-opt 時，每次的 input 都是上一次 iteration 的結果，因此沒辦法做 streaming。但若修改成在執行不同 start/end points 的部分做 streaming 的話是可行的，利用此方法應能有效的提升 utilization。
另外之前 homework 使用 streaming 無法得到更好的答案應該是因為 gpu 的資源都已經被佔滿，因此沒有辦法獲得速度上的提升。

About this course、

在這學期的課程裡學到了許多有用的知識，受益良多。感謝教授及助教們的用心，提供了相當棒的教材以及良好的寫作業環境。近日疫情嚴峻，祝福教授與助教們平安的度過本次疫情。非常感謝本學期的悉心教導。