Parallel Programming

Homework 3 Report: All-Pairs Shortest Path (CPU)

108062605 呂宸漢

1. Implementation

openMP：hw3.cc

a. 我選擇用 Floyd-Warshall Algorithm 解決 all-pairs shortest path 的問題，雖然他在計算的複雜度上可能會輸給 Dijkstra Algorithm，可是他在平行化的方面較容易實作，且比較沒有 data dependency 的問題。

b. 時間複雜度是 $V*O(V^2/P)$ (*V*: number of vertices, *P*: number of CPUs)，因為 Floyd-Warshall Algorithm 最外面的迴圈會有 data dependency 的問題，因此只能平行裡面兩個迴圈，雖然也會有 data dependency 的問題，不過不會影響答案的正確性，也會有機會提前降低 edge 的 distance，是好的 data dependency。

c. 因為這次不限制方法解決 all-pairs shortest path，考慮有人使用 Dijkstra 或是 Bellman-Ford 執行 V 次解決 all-pairs shortest path 的問題，以 Dijkstra 而言時間複雜度是 $V*O(E+V*logV)$，也就是 $O(V*E+V^2*logV)$，而 Bellman-Ford 的時間複雜度則是 $V*O(V*E)$，也就是 $O(V^2*E)$，而 Floyd-Warshall 的時間複雜度則是 $O(V^3)$，當 edge 少的時候，Dijkstra 可以有較好的 performance，可是當 edge 數為 V*(V-1)時，Floyd-Warshall 才會展現出它的優勢，因此我的 testcase 將 vertex 數量調到最大且 edge 數為 V*(V-1)，盡量加長運算與判斷的時間，當有人使用 Dijkstra 或 Bellman-Ford 時 runtime 則會被拉長，藉此增加超時的機率。

d. 讀取二進為檔案的方式則是用 ifstream，寫入則是用 ofstream，當開啟檔案時用 binary 開啟，並用 read 及 write 讀取和寫入，由於檔案內容都是 integer，所以是每 4 個 byte 為一組讀取和寫入，這也是一開始寫作業時遇到的第一個問題。

2. Experiment & Analysis

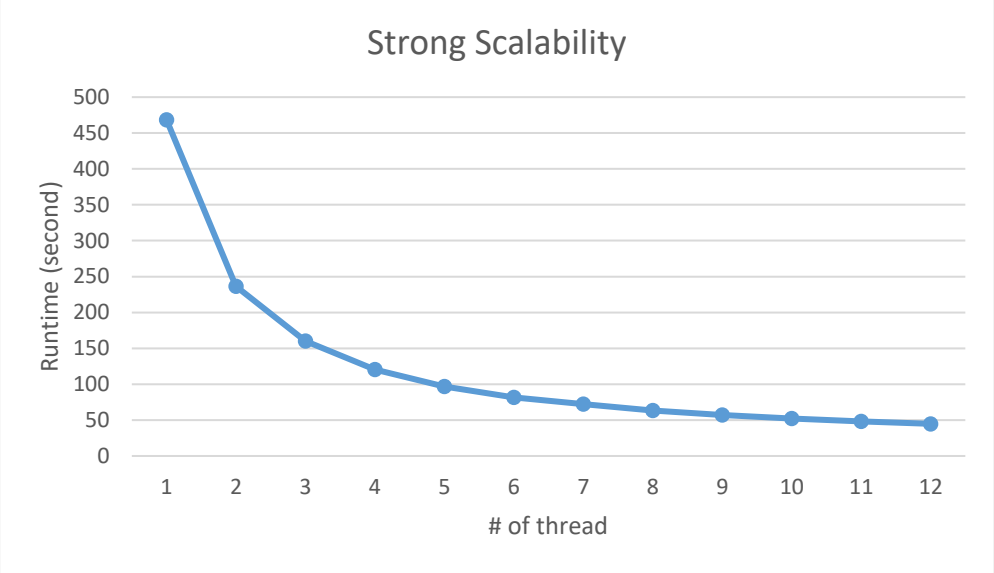
a. System Spec

所有程式皆在課程所提供的 cluster 上進行測試。

b. Strong Scalability

利用 omp_get_wtime()加在 main 函式的開頭和結尾，再相減即可得到程式的執行時間。

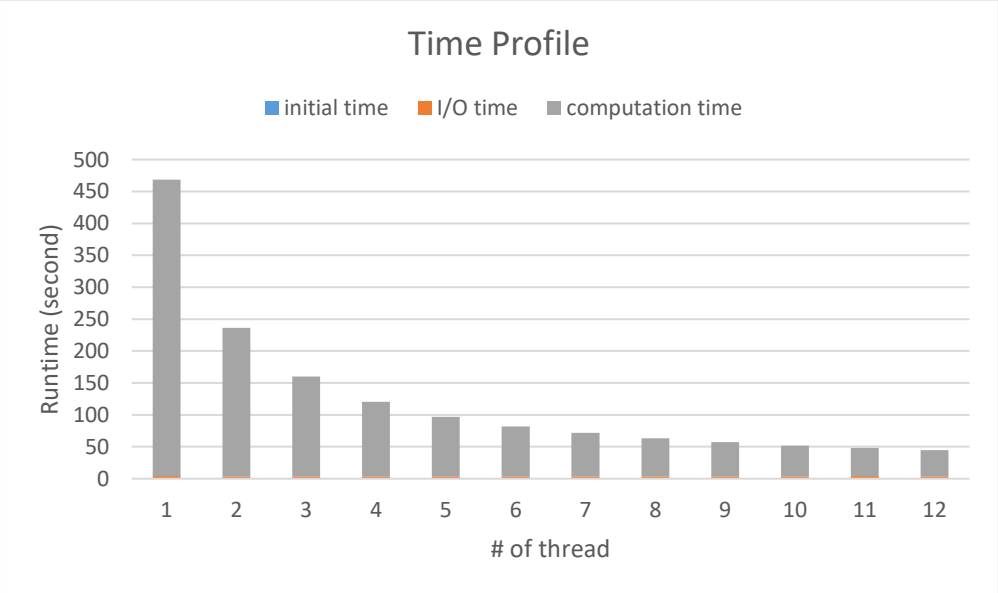
由下圖可見，thread 數增加則 runtime 隨之減少，不過在 9 條 thread 之後下降的幅度就漸緩了，我認為是因為 thread 管理佔據了一部份的時間，雖然 openMP 會自己做 thread 管理，可是仍然有一定的管理成本，尤其越多 thread 在分配資源時就會形成 bottleneck，才形成這樣的曲線。



c. Time Profile

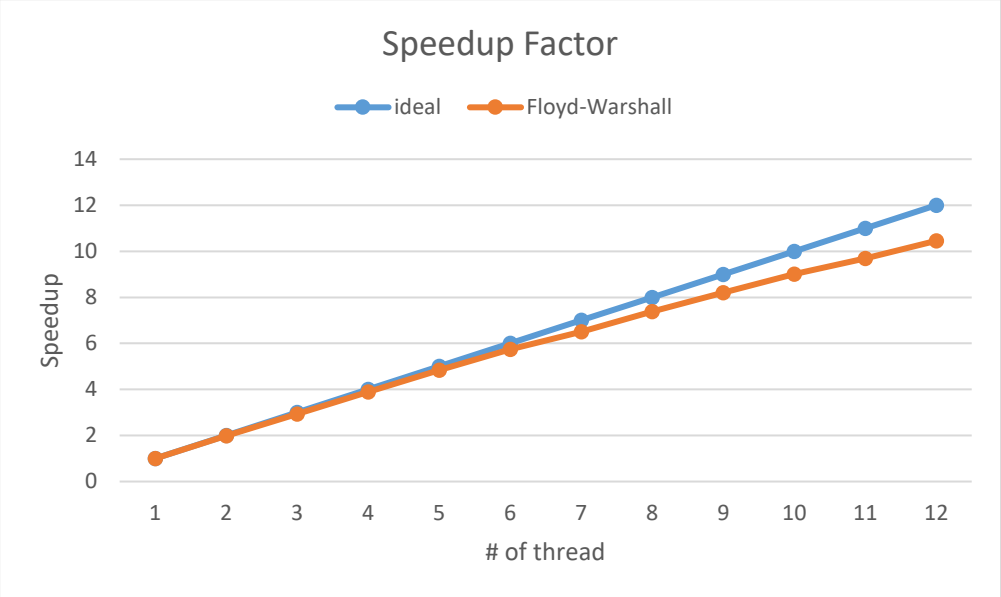
在 I/O 及 computation 的部分前後加上 omp_get_wtime()，再計算差值即可得到 I/O time 及 computation time。另外再用 total runtime 減掉 I/O 及 computation 即可得到 initial data structure 的時間。

由下圖可見，由於 initial time 及 I/O time 太小，導致在圖中看不太出他們的時間，computation time 的結論則與 strong scalability 的結論相同。



d. Speedup Factor

這是我額外做的圖表，用以輔助說明 strong scalability 的結論，由下圖的資料線可以看出，在小於 6 條 thread 時，此程式的 speedup 與理想的狀況大致相同，不過在大於 6 條 thread 後，與理想曲線的差距就會隨的 thread 數增加而越大，表示在 thread 數變多後，加速的效率就會降低，如同在 strong scalability 的結論，因為 thread 數變多，管理 thread 需要花費更多時間，才會造成這種現象。



3. Experiences / Conclusion

由於期中的作業較多，時間分配的不恰當，導致這次作業分配到的時間被壓縮了許多，沒能像之前幾次作業花費較多的時間在 speedup 上，尤其這次還要自己創造 testcase，而且評分方式也不同，如果有更多的時間可以寫，或許可以再想到更多加速的方法，當知道加速的方法後，也會有比較多的資訊可以考量，在創造 testcase 時可以有更多的想法拖慢執行速度，最 general 的想法應該都是把 vertex 與 edge 數調到最大，增加需要計算的的數據量及 I/O 的時間。很遺憾可以用在這次作業的時間太少，導致學習到的東西不多，報告也比較少東西可以寫，下次必須好好分配時間，再為下個作業努力。