Homework3: All Pair Shortest Path 羅允辰 103061108

Implementation

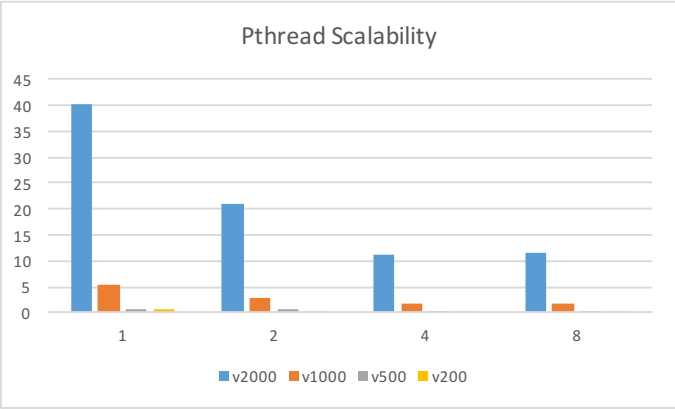
- Pthread Version Detail:
  1. I use Floyd Warshall algorithm(Because this algorithm can be implemented by only three for loops, it is very simple for parallel computing compared to Dijkstra's algorithm with the same time complexity)
  2. I break the inner two for loop for parallelization. Since the outmost loop cannot be parallelized because of data dependency. I first collapse the inner two for loops for easier implementation.
- Synchronous MPI Version Detail:
  1. pros: Easier to determine termination condition.
  2. cons: It need a collective call to ensure the termination condition, when the process number is extremely large, ex, 10000. This kind of collective call will be extremely expensive.
  3. I use all reduce to gather the termination condition.
- Asynchronous MPI Version Detail: Use collapse and parallelize each pixel!
  1. pros: Don't need a master program to check if the collective call satisfying termination condition or not.
  2. cons: Need more channel to pass the token, and harder to implement.
- Additional Effort:
  1. Implement Dynamic Programming to neglect mighty duplicated iteration on both MPI version.
  2. implement 2 versions of read all activation signal and compare.

Experiment & Analysis

I. Methodology
   (a) System Spec: provided by TAs
   (b) Performance Metrics:
      1. Communication time: 使用 MPI_Wtime 將有傳輸的地方包住
      2. IO time, Computing Time: 同上

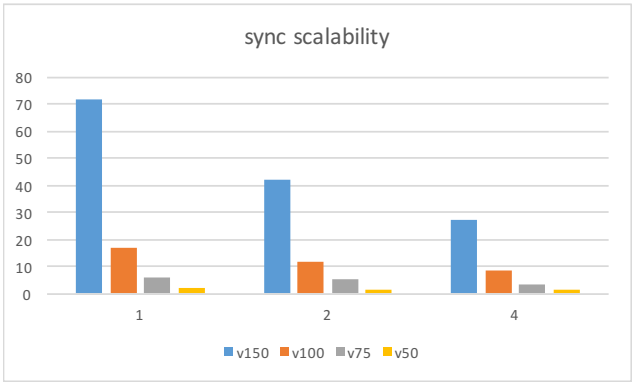
II. Time Profile
   (a) Strong Scalability Chart
      (i) Pthread

我們可以觀察到當 thread number 到達 4 以上以後，即便是最大的測資(v2000)，已經沒有辦法再進一步加速，而原因則是因為 thread creation time 所導致的。



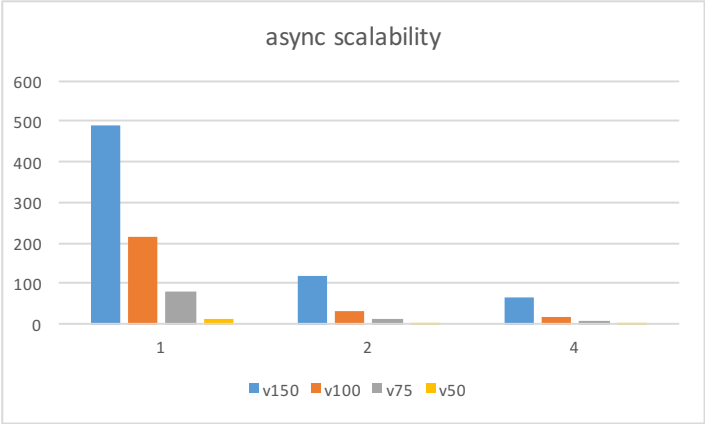
(ii) MPI sync(x: number of real nodes, y: s)

我們可以觀察到當 real core 增加以後，實際的運行速度也會大幅下降，我認為這是因為，他的運算單元變多時，可以讓我們一些 mpi call 的能更快執行完畢所致。



(iii) MPI async(x: number of real nodes, y: s)

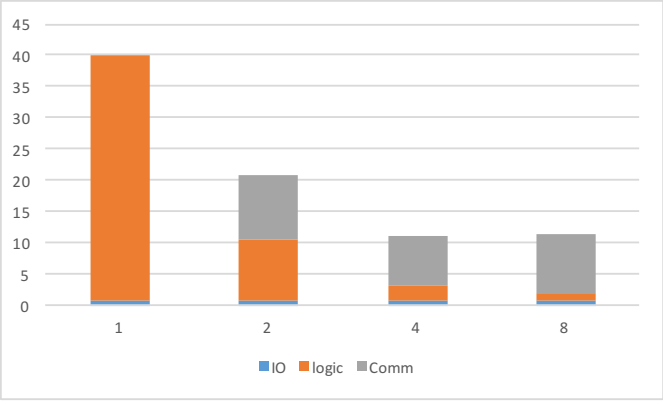
我們可以觀察到當 real core 增加以後，實際的運行速度也會大幅下降同。
我認為這跟 sync 很類似，同樣也是 mpi 的某些 function call 因為運算資源變多，可以更快地執行完畢不會卡住。



(b) Performance Profiling

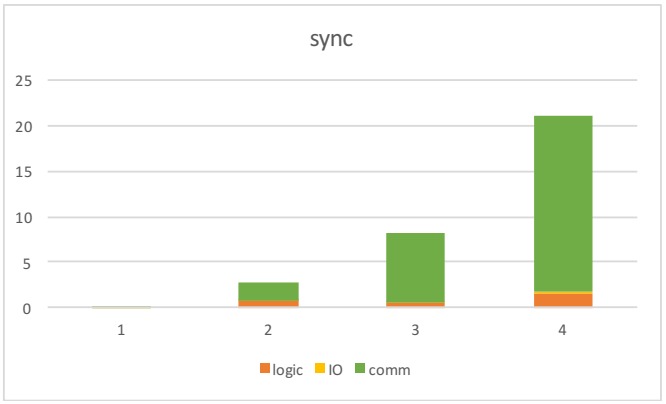
(i) Pthread

我們可以觀察到當 thread number 到達 4 以上以後，因為創造 threads 的 overhead 已經變得很大，所以真正變成主要花費時間的 bottleneck 已經成為 thread creation 的 overhead!



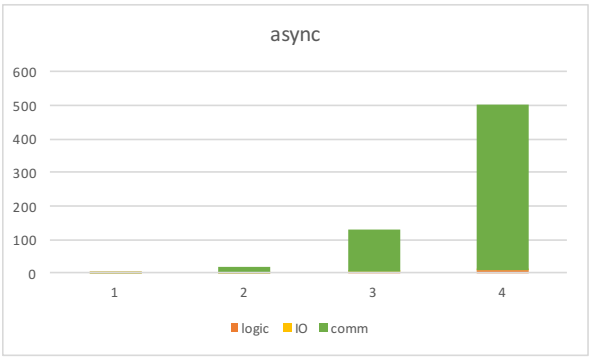
(ii) MPI sync(x: different testcase , y: s)

我們可以發現，在右圖當中，雖然 comm time 都是最多的，但是我認為有些是 IDLE 的時間而不是 communication time，最重要的是這樣跑的話主要的時間都會花在 communication 而不是計算，所以是不太好的平行處理法。
TEST CASE1(50 VERTEX)
TEST CASE2(100 VERTEX)
TEST CASE3(150 VERTEX)
TEST CASE4(200 VERTEX)

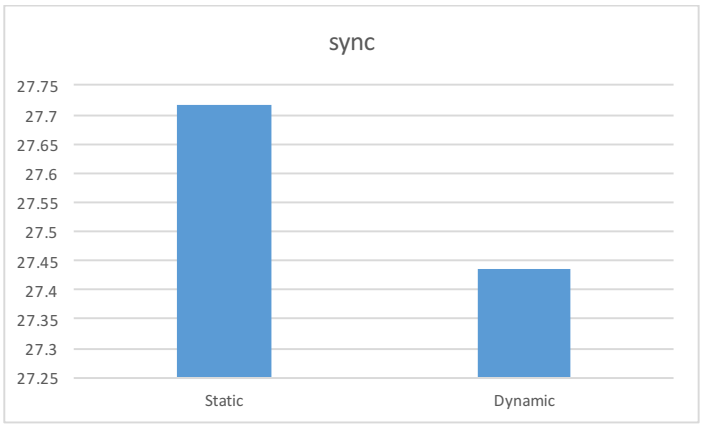


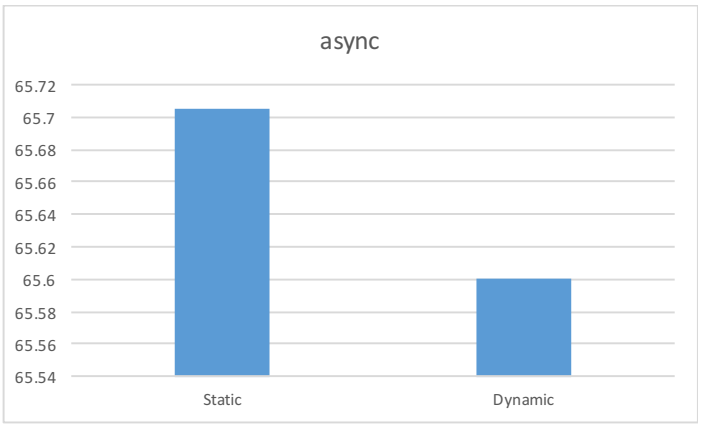
(iii) MPI async(x: different testcases, y: s)

類似於 sync 的版本，隨著 vertex 的變多時間會變呈倍數成長，但是不變的是主要的時間都是 communication time
但是，使用 dual ring 的 termination condition，執行時間會慢上超級多，所以才有 hybrid 的版本出現的必要性。(但是沒有時間做 QQ)
Test case 順序同 sync



(c) Dynamic Programming compared to Iterative-only version(MPI)





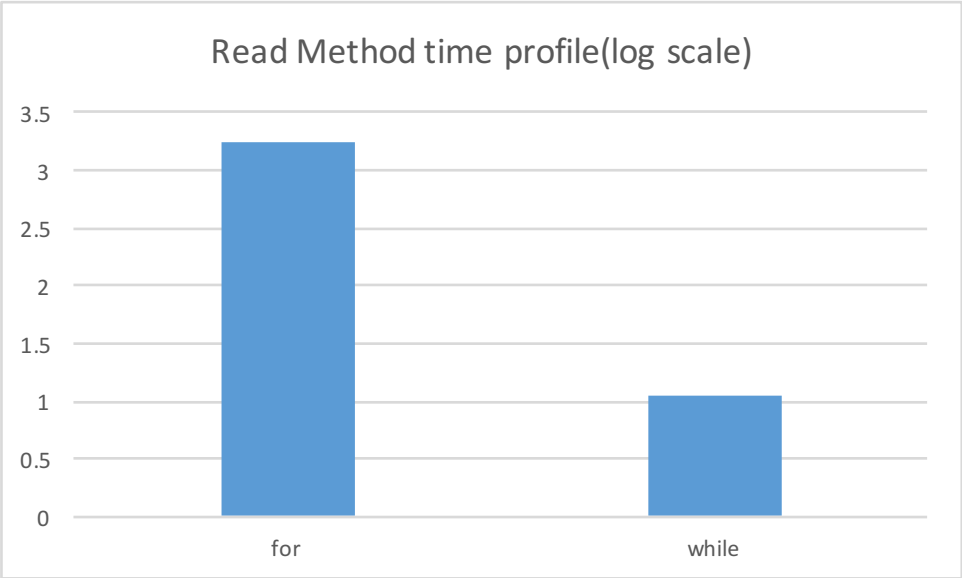
上兩張圖都可以讓我們發現，當使用 dynamic programming 的技巧，將已經算過的 distance 先存起來不用在算第二遍，可以節省 iterative 的時間，利用 adjancent matrix 第

(i, j)格的距離應該要跟 (j, i)一樣的特性，將在算完其中一個 iteration 的map 複製到新的 map 上，希望能減少新的 iteration 所花的時間，但因為在實作中時間主要並非花費在 logic 上，所以加速並不顯著。

額外實驗發現：

1. 剛開始，上面的實驗當中，由於使用 Moore's Algorithm 時，會是逐步以每個 vertex 作為 root 來找他到每一個節點的最短路徑，在 Moore's Algorithm 中，會有可能發生很多個 vertex 中同時傳給同一個 vertex 的狀況，因此，我們必須要實作 finite state machine 的機制，將每一次的 iteration 分成 Probe+Read, judge if need to activate others, Send signal to activate others。在這之中，必定要利用 Isend 所會產生的 buffer 來讓 Read 時可以逐步消化很多人傳給他的訊息，而在Read 上，我實做了兩種機制，如下分別描述，並分析優點及帶來的問題。
    1. while 實作方法：當使用這 while 來實做時，會 iprobe 任何想要傳進來的 process ，並且在沒有人想要送之後就停下來，雖然這個實作方法照理來說會比 for 的實作方法來的快，實際在一個 node 上測試也是如此，但是卻無法成功的在多個 node 上 run 出正確的值。我最後試了很多作法，但是卻都無法解掉這個問題(我也是黑人問號 QQ)
    2. For 實作方法 ：使用 for loop 來 probe 的話，會逐步地按造順序來 probe 所有的 process，當有發現需要讀進來的訊息時，就會 bocking read 將東西讀進來，但是這個做法卻一定要按造順序 probe 所有的人一遍，有可能會造成不必要的 probe ，雖然這個實作方法比 while 的實作方法來得慢，但是卻可以確定 run 在多個 node 下都一直是正確的，因此我最後選擇 run scalibility 的演算法為這個演算法。

       下表的時間為 log scale 的時間！



Experience/ Conclusion

1. This homework is very hard! But with the implementation on distributed version of parallel code, I am able to understand the methodology of creating a fully-distributed version of parallel code!
2. The hardest parts of the implementation are to find the suitable algorithm and the condition in MPI that many vertexes may send to the same vertex at the same time!