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.
- Additional Effort:
    1. Implement Dynamic Programming to neglect mighty duplicated iteration on both MPI version.

Experiment & Analysis

I. Methodology

(a) System Spec: provided by TAs

(b) Performance Metrics:

1. Communication time:
2. IO time:
3. Computing time:

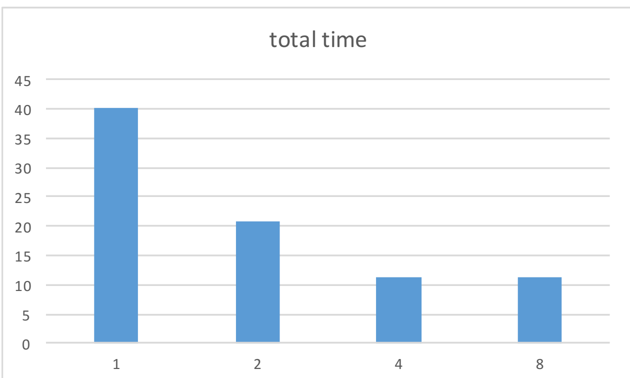
(c) The test data for pthread I choose is Weighted Graph with 2000 vertexes & 4000000 edges.

II. Time Profile

(a) Strong Scalability Chart

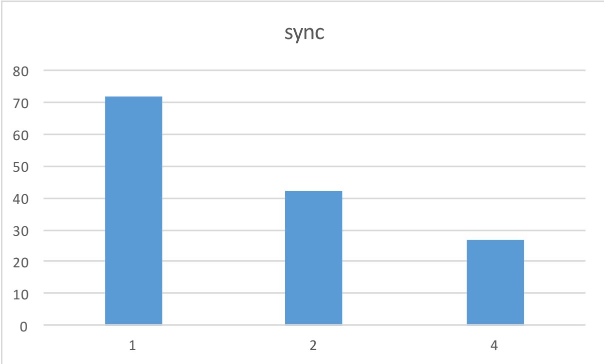
(i) Pthread

我們可以觀察到當 thread number 到達 4 以上以後，每個已經沒有辦法再進一步加速，而原因將於下段圖表呈現。



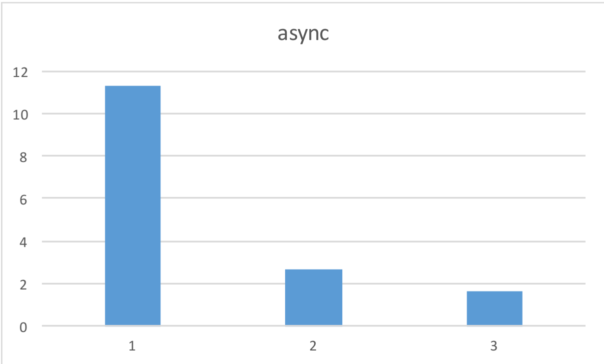
(ii) MPI sync(total vertex: 150, x: number of real nodes, y: s)

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。



(iii) MPI async(total vertex: 50, x: number of real nodes, y: s)

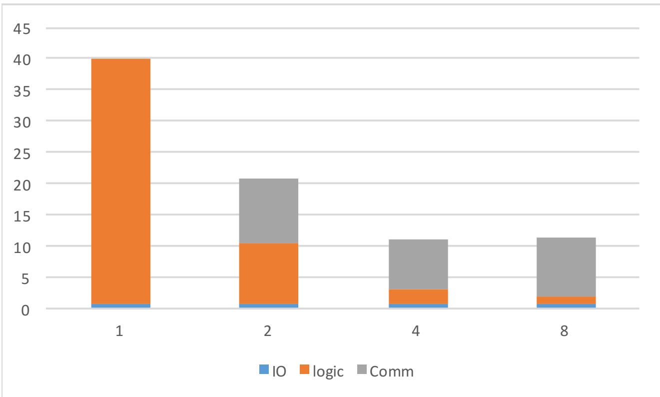
我們可以觀察到當 real core 增加以後，實際的運行速度也會大幅下降。



(b) Performance Profiling

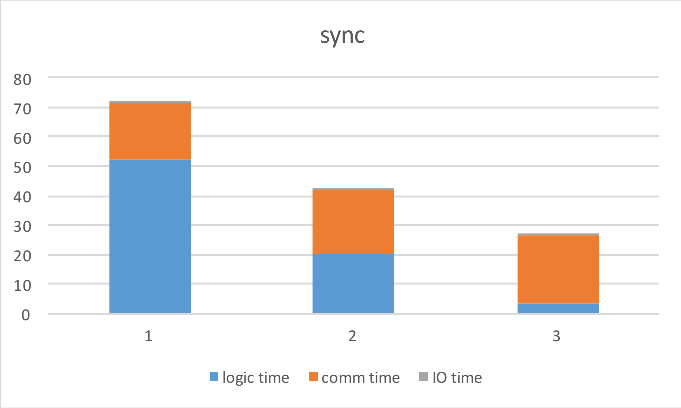
(i) Pthread

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已經成為 thread creation 的 overhead!



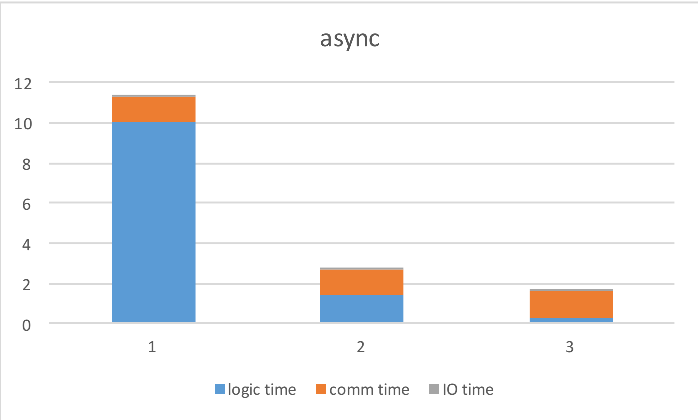
(ii) MPI sync(test case same as scalability experiment)

我們可以發現，在右圖當中，雖然 logic time 都是最多的，但是因為我們 logic time 的計算方法為扣除 communication 的時間，所以也包含 idle 的時間，若我們的 core 越多實際上也可以跑得更快。

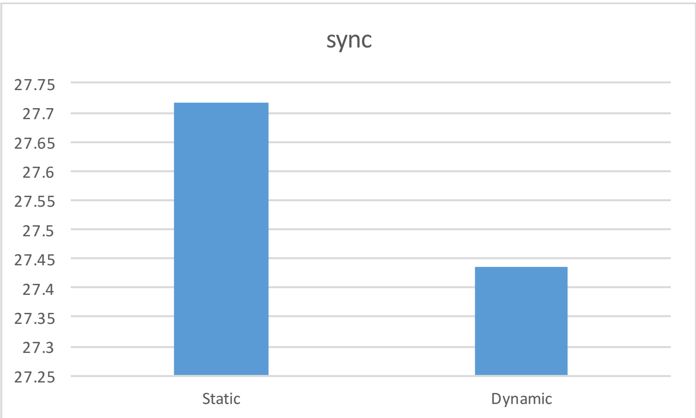


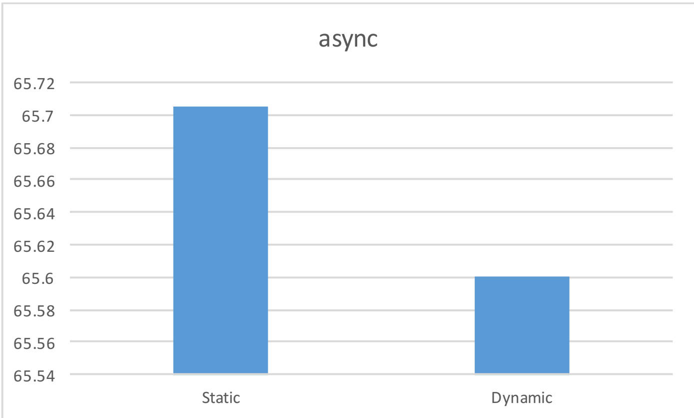
(iii) MPI async(test case same as scalability experiment)

我們可以發現，在右圖當中，雖然 logic time 都是最多的，但是因為我們 logic time 的計算方法為扣除 communication 的時間，所以也包含 idle 的時間，若我們的 core 越多實際上也可以跑得更快。我認為這是因為他不用再多 context switch 或是 idle 的關係。



(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，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。

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!