Final Project: Maximum Clique Problem
Group 7

1 Research topics

本組的研究題目為利用 MPI 及 OpenMP 加速解決 Maximum Clique Problem。給定一個 undirected graph G, 若存在一 subgraph，此 subgraph 中所有的 vertex 兩兩之間皆存在 edge，則稱此 subgraph 為 G 的一個 clique。而 Maximum Clique Problem 就是在 undirected graph G 中尋找一個 vertex 數量最多的 clique。此問題同時也是一個 NP-complete 問題。

2 Research motivation

先前在修資料結構及演算法課程時，皆有提到 Maximum Clique Problem，那時便對這個問題產生了極大的興趣。因此，藉由這次的 Final Project，希望能夠好好地了解這個問題，並利用平行計算的方法實作出比原本更快的解決方法。

3 Research method

(1)、在查閱許多與 Maximum Clique Problem 相關的論文資料後，本組選擇了較易平行化的 Bron–Kerbosch algorithm 來進行接下來的實作及研究。
(2)、實作 Bron–Kerbosch algorithm 的 baseline 版本，產生測資驗證程式實作的正確性，並與 boost library 中的 bron_kerbosch API 進行對比。
(3)、實作 Bron–Kerbosch algorithm 的 pivot 版本，與先前的版本進行比較。
(4)、使用 OpenMP 對程式進行優化，並與先前的版本進行比較。
(5)、使用 MPI 及 OpenMP 對程式進行優化，並與先前的版本進行比較。

4 Implementation

(1)、Bron–Kerbosch algorithm:

```
algorithm BronKerbosch1(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    for each vertex v in P do
        BronKerbosch1(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

簡單來說，Bron–Kerbosch algorithm 就是對每一個 vertex 維持一個 maximal clique 的 set，尋找兩兩之間有 edge 相連的 vertex 加入 set 中，直到最後找不到能夠加到 set 的 vertex 之後，就找到了一個 maximal clique。

(2)、Bron–Kerbosch algorithm with pivot:

```
algorithm BronKerbosch2(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Bron–Kerbosch algorithm 會有許多 vertex 被重複計算很多次。舉例來說，vertex A 與 B 之間有 edge 相連，BK algorithm 會先尋找包含 A 的最大 clique，接著會尋找包含 B 的最大 clique。因為 A、B 相連的關係，因此 set(A, B)及 set(B, A)就會被重複計算。使用 pivot 能夠將重複計算的部分刪除，讓演算法更有效率。

(3)、OpenMP version:

```
algorithm BronKerbosch2(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

因為在 Bron–Kerbosch algorithm 中，每個 vertex 都會維持自己獨立的一個 maximal clique set，因此最直覺的平行方法就是讓計算一個 vertex 的 maximal clique set 為單位任務量，接著用多個 CPU 去平行計算這些任務。對應演算法就是把第一層的 for loop 使用 OpenMP 平行。

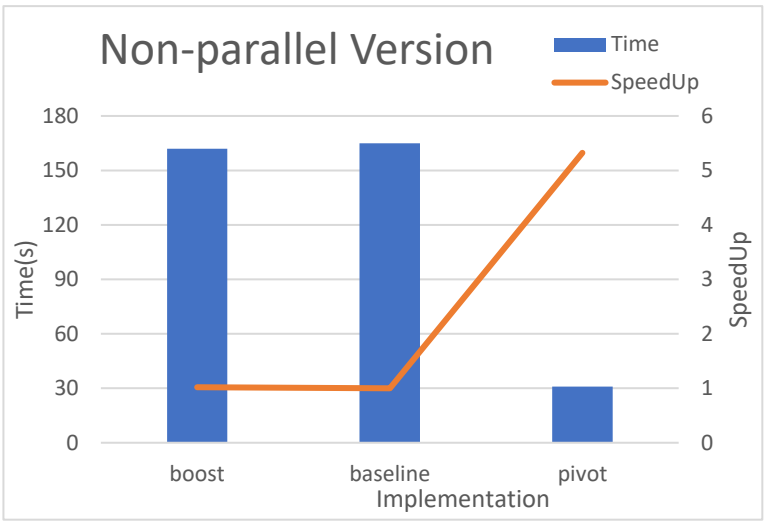
(4)、MPI+OpenMP version:

```
algorithm BronKerbosch2(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

在考慮多個 node 後，平行的方式也可以變成 hierarchical 的方式進行實作。主要就是讓其中一個 node 的一個 CPU 作為 master，維持計算任務的 work pool，work pool 中為第一層的 for loop 的計算量；當其他的 node 收到第一層的計算任務，變將第二層的 for loop 利用 OpenMP 進行平行。

5 Experiment & Analysis

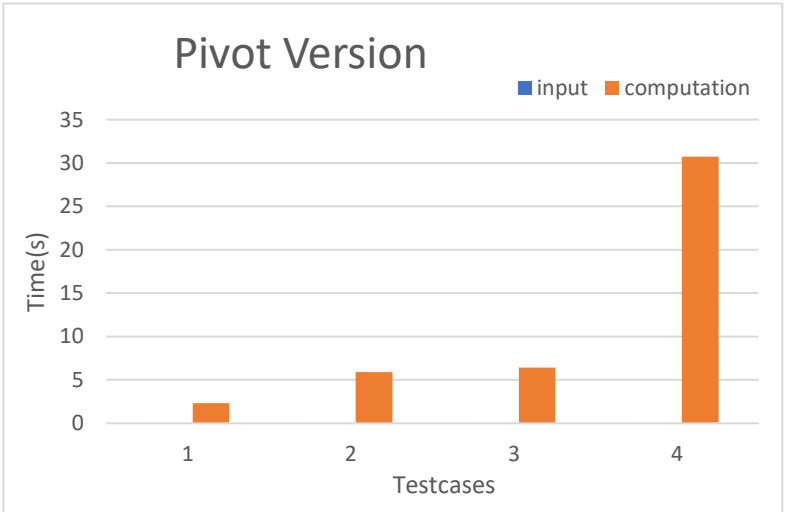
(1)、未經平行的版本時間分析(boost、baseline、pivot)



分析：

a)、boost library 版本和未經優化的 baseline 版本時間差不多，而 pivot 版本則比 baseline 加快了至少 5 倍，因為 pivot 版本大量減少了程式中 call recursive 的次數。因此以 pivot 版本為基礎進行接下來的實作。

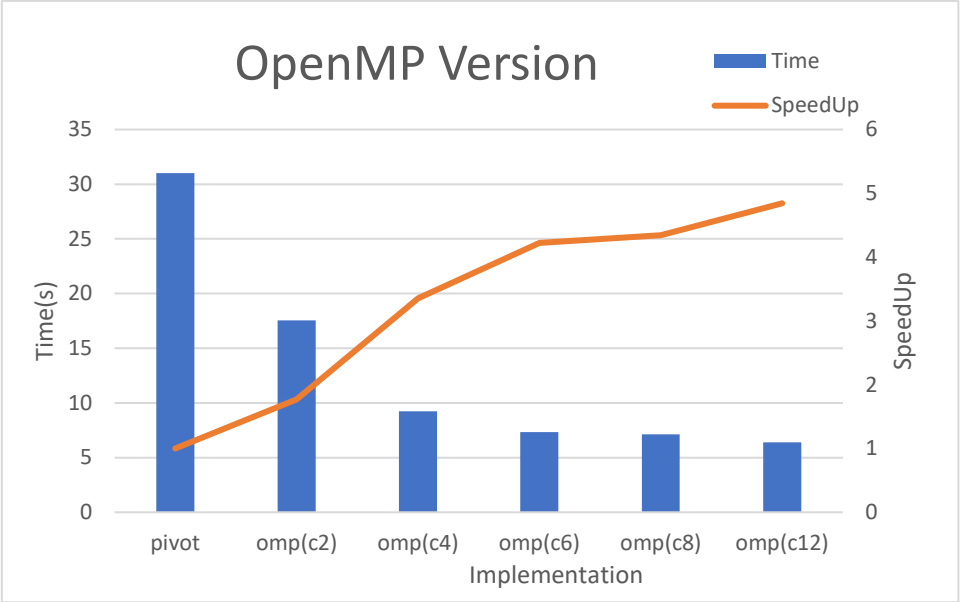
(2)、pivot 版本 time profile 分析



分析：

a)、input time 相較於 computation time 基本上可以忽略不計，因此 bottleneck 為 computation time。

(3)、OpenMP 版本時間分析(以 pivot 版本為 base)

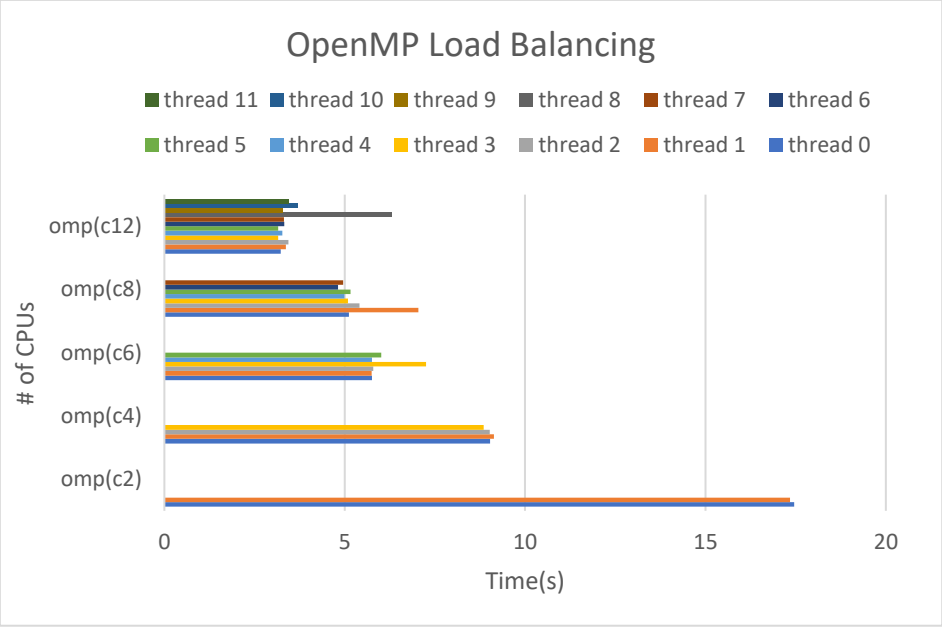


分析：

a)、由圖可得，OpenMP 確實能夠加速原本沒有平行的版本。

b)、隨著 CPU 數量增加，時間減少量越來越小，加速也漸漸趨緩，像是當 CPU 數為 6、8、12，三者時間幾乎一樣，scalability 並沒有很好。

(4)、OpenMP 版本 load balancing 分析

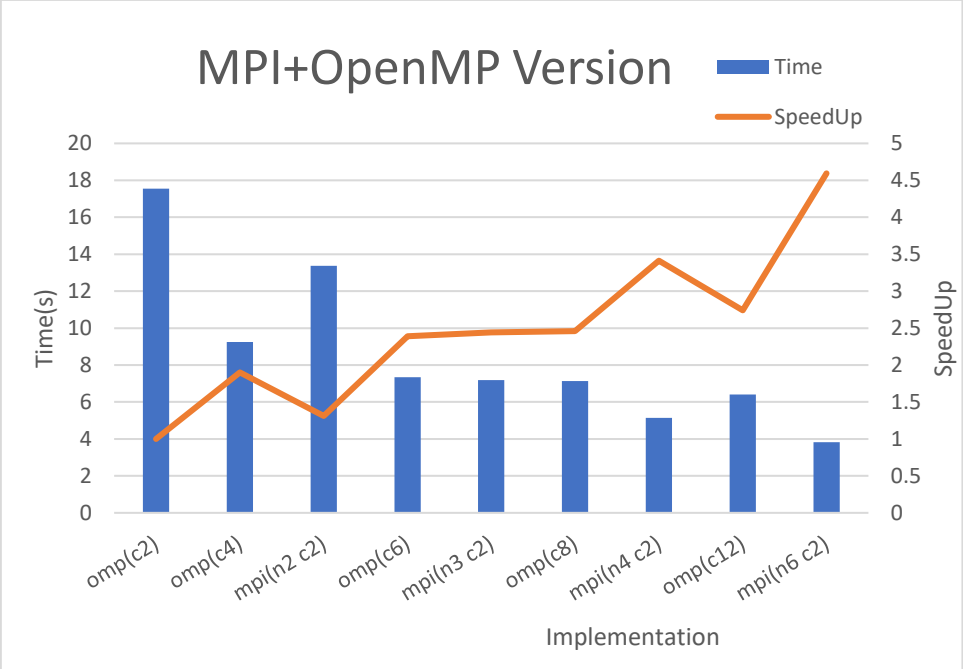


分析：

a)、當 CPU 數量增加的時候，基本上每個 CPU 所分到的任務量會減少，計算使用的時間也會減少。

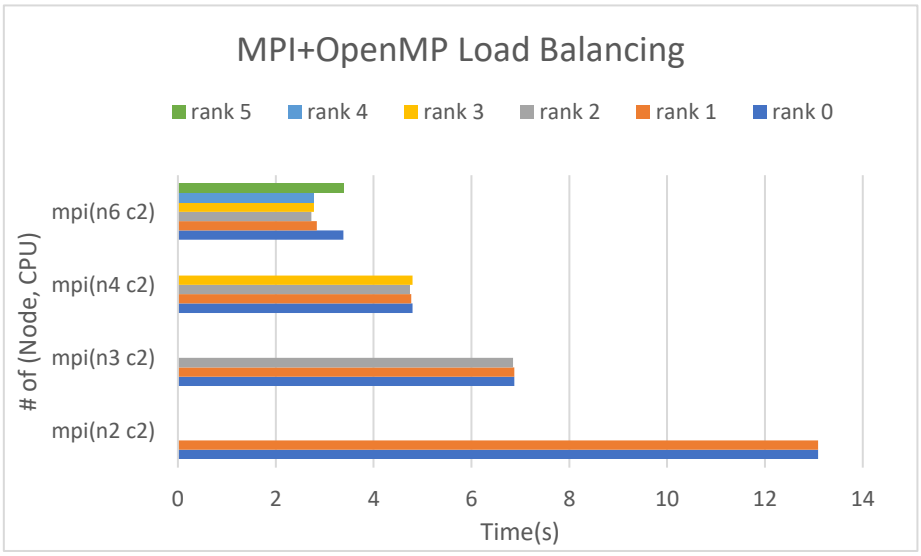
b)、在 CPU 數量為 6、8、12 時，大部分的 CPU 所計算的時間降低，唯有其中某一個 CPU 計算時間仍遠高於其它 CPU，導致了 CPU 數量為 6、8、12 時計算時間相差不大。
c)、因為無法預測單位任務量的大小(跟 edge 的分布有關)，當 CPU 所分到的任務量變少的時候，某些計算量較大的任務就會成為 bottleneck。

(5)、MPI+OpenMP 版本時間分析(以 omp(c2)版本為 base)



分析：
a)、以(node x CPU)數量為計算單元的數量進行比較。
b)、當 node 數只有兩個時，因為 master rank 需要負責分配任務及計算，CPU 數也只有兩個，因此所用時間比使用 4 個 CPU 的 OpenMP 版本慢。
c)、當 node 數增加之後，慢慢會比用同等數量計算單元的 OpenMP 版本要快，得益於 MPI 平行了第一層的計算，OpenMP 平行了第二層的計算，相較於 OpenMP 版本將所有 CPU 皆用於平行第一層，可以預期，平行兩層會有更好的 load balance。

(6)、MPI+OpenMP 版本 load balancing 分析

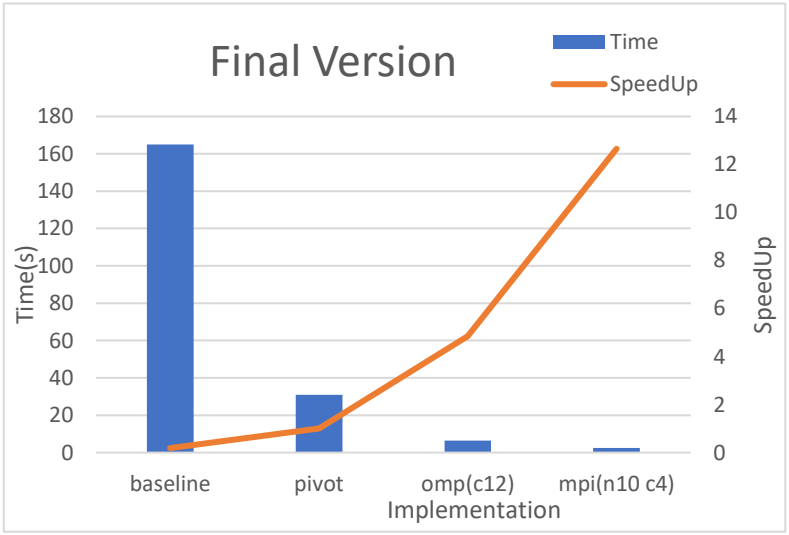


分析：

a)、在同等計算單元的情況下，相較於 OpenMP 版本只平行第一層，平行兩層的 MPI+OpenMP 版本 load balance 更好，因此擁有更好的平行計算的效能。

b)、在 node 數繼續增加之後，也會因為平行第一層的關係導致 loading 失去平衡。

(7)、最終版本時間比較(以 pivot 版本為 base)



分析：

a)、以沒有平行的 pivot 版本為 base，使用 OpenMP(c12)可以加快將近 5 倍；使用 MPI+OpenMP(n10 c4)可以加快 12 倍。

7 Conclusion

(1)、Maximum Clique Problem 是很經典的 NP-complete 問題，Bron–Kerbosch algorithm 是其中比較簡單、比較容易平行化的一種解決方法，它並不是最快的解決方法。

(2)、平行計算是將計算平分給許多 node、CPU 一起執行，但當計算的時間複雜度成指數倍上升，且計算量很大時，有時平行計算也可能不足以解決這個問題。