國 立 交 通 大 學

統計學研究所

碩 士 論 文

對 K 均值分群估計潛在群體程序作平行運算

Parallel Computing for K-means Clustering on Estimating Underlying Latent Classes

研 究 生：林吟玲

指導教授：黃冠華　博士

中 華 民 國 九 十 九 年 六 月

對 K 均值分群估計潛在群體程序作平行運算

Parallel Computing for K-means Clustering on Estimating Underlying Latent Classes

研 究 生：林吟玲　　Student: Yin-Ling Lin

指導教授：黃冠華　　Advisor: Dr. Guan-Hua Huang

A Thesis Submitted to
institute of Statistics
College of Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Statistics
June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年六月

對 K 均值分群估計潛在群體程序作平行運算

研究生：林吟玲　　指導教授：黃冠華　博士

國立交通大學統計學研究所

摘要

本論文主要目的是對 k-means 分群方法估計潛在群體的計算過程作平行運算，透過 OpenMP 與 MPI 平行運算，將 updated k-means 與 non-updated k-means 兩種不同 k-means 分群方法作平行，使得程式計算時間縮短，並且在個人電腦、國家高速電腦中心與 Amazon EC2 三種不同電腦環境上運作，觀測他們的平行效率。除此之外，利用乳癌的微陣列為例，作更詳細的說明。在乳癌資料的例子中，兩種 k-means 分群方法都達到縮短運算時間的效果！

關鍵字：OpenMP，MPI，平行運算。

Parallel Computing for K-means Clustering on Estimating Underlying Latent Classes

Student: Yin-Ling Lin Advisor: Dr. Guan-Hua Huang

Institute of Statistics
National Chiao Tung University

ABSTRACT

The main purpose of the study is to perform parallel computing for k-means clustering on estimating the underlying latent class process. OpenMP and MPI parallel computing make computing time shorter for updated and non-updated k-means clustering method. We compare the parallel efficiency of OpenMP and MPI in the personal computers, the national center for high-performance computing and the Amazon EC2 environment. Besides, the breast cancer microarray data are used for illustration. The results display that parallel computing can reduce the computation time in all three computing environments.

Key words：OpenMP，MPI，Parallel Cmputing.

誌謝

最想要感謝的是我的指導教授黃冠華老師，這一年老師引導我如何去解決問題並且養成良好的學習態度，使得我能順利的完成論文，同時感謝老師幫我找到中研院的研究工作，讓我有一個好的環境能繼續學習；也感謝其他老師在課堂上的教導，使我用不同的觀點去看待事情，使我有所成長。

接著謝謝這兩年來一起修課的好夥伴弘哲，因為有你的督促，使我能如期完成許多事情；謝謝我的室友們郁涵與千慧，因為有你們的陪伴，使我碩二這年過得很愉快；謝謝 408 研究室的所有同學，一起在大桌子聊天、吐苦水的畫面，是我最懷念的時光；謝謝所辦的郭姐，因為有你幫我們處理大大小小的所務，所上的大家才能專心的做自己的研究，感謝所上每一位老師、同學、學長姐、學弟妹，因為有你們的照顧，使我這兩年來，不僅得到很多知識，也擁有深厚的友誼！

感謝黃冠華老師、鄭又仁老師、陳君厚老師、陳鄰安老師，因為有你們寶貴的意見，使我的論文更加完善。

最後感謝支持我的家人與朋友們，謝謝你們這兩年的支持與鼓勵，讓我無後顧之憂的朝著自己的理想邁進。

林吟玲 謹 誌于

國立交通大學統計研究所

中華民國九十九年六月二十三日

目錄

表目錄

圖目錄

1. 介紹

潛在變量模型(Latent variable models)是用於解釋不可觀測的變量和他們的觀測值之間關係。潛在變量模型假定所有觀測值反映了同樣的不可觀察之特點，且其特點充分說明了觀測值之間的關聯。這不可觀測的特徵可以表示為一個單變量連續變數（潛在特質）(Rasch, 1960; Lazarsfeld and Henry, 1968; Moustaki, 1996)或群體變數確定數個“種類”去定義同一群體中的樣本(Goodman,1974; Titterington and others, 1985; Bandeen-Roche and others, 1997; Huang and Bandeen-Roche,2004)。我們著重於觀察潛在群體變數，使得在任何潛在群體變數內的觀測值互相獨立，其稱為潛在群體模型(Latent class analysis)。

在固定個數群體的潛在群體模型通常是由最大概似(maximum likelihood)去做參數估計。因為不能觀察群體中的成員，使得潛在群體模型成為一個典型不完整數據的問題，若使用EM演算法(Expectation-Maximization algorithm)(Dempster and others, 1977)計算最大概似所估計的參數，既不能直接驗算，而且有可能嚴重影響分析結果(Bandeen-Roche and others, 1997)。此外，使用EM演算法來估計模型參數，通常是費時且在樣本個數小於觀測值時難以收斂。

潛在群體模型分析通常被視為與群體分析(cluster analysis)相似，因此我們使用群體分析來估計潛在群體(Huang, Wang &Hsu, manuscript)。群體分析包含不同種演算法，其方法都是將相似(有較接近的距離)的物體分別納入同一群體中。使用k-means的演算法，將原本的距離測度改成相關係數或共變異數，然後對所有的主體分群，使得屬於在同一群的主體所測得的觀測值能相互獨立，再將估計出的潛在群體視為已知變數後，就很容易在潛在群體模型估計參數。該方法可以很容易地處理高維度的數據，並可直接獲得最佳的潛在群體估計。

而在處理高維度數據時，常常會遇到記憶體不足或者是運算時間過久的問題，若遇到記憶體不足就必須要在電腦的硬體上做加強，(例如:增加運算的記憶體空間或者借用硬碟上的記憶體);若遇到運算過久則要在程式設計上做處理，(例如:尋找更快的運算方法或者加入平行運算設計)。

為了加強電腦硬體設備，常常需要花費大量的金錢且需要人力去維修，且不想使用又不能丟棄，因此在本論文中我們選擇使用 Amazon EC2

的雲端服務，隨時增加或減少當下所需要的電腦設備，一來減少金錢與人力支出上的耗費，二來虛擬電腦也不占空間，達到按需即取的好處。

在增加程式運算效率的方法上，我們加入平行運算的架構，在多核心的電腦上使用 OpenMP 平行運算語言，在電腦叢集環境中使用 MPI 平行運算語言，以減少運算時間上的耗費。

本論文將利用潛在群體模型估計潛在群體的程式來探討 OpenMP 與 MPI 平行運算語言的效果，且分別運行在個人電腦、國網中心以及 Amaozn EC2 虛擬電腦上。

2. k-means 分群方法

我們在 k-means 分群方法中引用兩種演算法，一種是在將資料重新分群時，其群體的中心值也隨之改變，另一種是在所有資料皆完成重新分群時，才將群體中心值重新計算。為了區分上的方便，我們稱前者演算法為 updated k-means 分群方法，稱後者演算法為 non-updated k-means 分群方法。此外，我們在對non-updated k-means分群方法平行時引用parallel k-means 分群方法(Kraj and others, 2008)的演算法概念。

2.1 Non-updated k-means 分群方法的演算法

nk1. 隨機將樣本分成 k 群且計算群體中心值。

nk2. 將每個樣本重新分配到與其最接近中心值的群體中，當所有樣本分配完成後，使用新的分群結果重新計算群體中心值。

nk3. 反覆執行 nk2 直到全部樣本皆沒有移動為止。

2.2 Updated k-means 分群方法的演算法

uk1. 隨機將樣本分成 k 群且計算初始群體中心值。

uk2. 將每個樣本重新分配到與其最接近中心值的群體中，當每個樣本被移動，群體中心值也將重新計算。

uk3. 反覆執行 uk2 直到全部樣本皆沒有移動為止。

2.3 Parallel k-means 分群方法的演算法

pk1. 隨機將 N 個觀測值分成 k 群且計算初始群體中心值，再將 N 個觀測值平均分成 M 個小觀測值體(S)。

pk2. 傳送小觀測值體(S)、群體個數(k)與初始群體中心值到 M 個電腦節點上。

pk3. 在每個電腦節點上，將小觀測值體中的每個樣本重新分配到與其最接近中心值的群體中。

pk4. 在每個電腦節點上，計算

$$FM_c = \sum_{j=1}^{m}\sum_{i=1}^{n_c} V_{ij} \quad , \quad SecM_c = \sum_{j=1}^{m}\sum_{i=1}^{n_c} V_{ij}^2$$

其中 c 為群體的指標，$c \in \{1,\cdots,k\}$，i 為觀測值的指標，j 為樣本的指標，m 為樣本的總個數，n_c 為觀測值的總個數。

pk5. 主電腦節點接收 FM_c、$SecM_c$ 與 n_c，並且計算

$$gCC_c = \frac{\sum_{p=0}^{M} FM_{cp}}{\sum_{p=0}^{M} n_{cp}} \quad , \quad Perf_c = SecM_c - \frac{FM_c}{n_c} \quad , \quad gPerf_c = \sum_{p=0}^{M} Perf_{cp}$$

其中 p 為電腦節點的指標，gCC_c 為新的群體中心值，$gPerf_c$ 為性能函數。

pk6. 主電腦節點傳送新的群體中心值給其他電腦節點，並且反覆執行 pk3 到 pk5 步驟直到性能函數達到最小值或者 m 個觀測值沒有被移動為止。

3. OpenMP 與 MPI 平行方法

3.1 平行概念

對於多核心或叢集環境並沒有辦法對一般程式提昇效能，但對於平行過後的程式，就可透過不同核心或多台電腦同時運算達到加速效果！

以一個簡單例子，沒有平行的程式，跑一次迴圈要 10 秒時間，且做 100 次都丟給同一顆核心做的話，要花費 10*100=1000 秒的時間，若以平行的程式來看，將這迴圈分給四顆核心做，每顆各做 25 次，則 只花費 250 秒的時間。

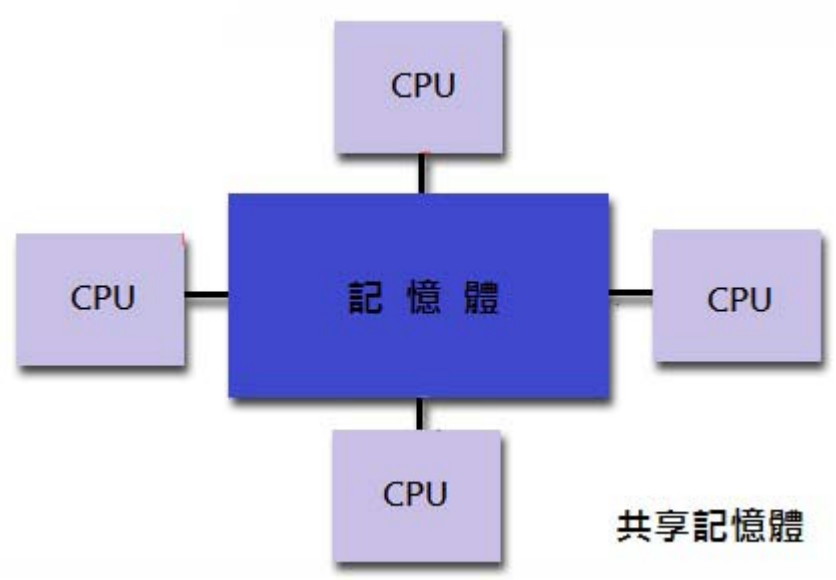
然而，平行的程式沒有想像中簡單，在工作切割與結合上，也是需要時間的，在叢集環境中還包含了資料傳輸所耗費的時間，除此之外，並不是所有程式都可以切割平行的！若程式具有相依性，直接用平行方式運算，結果是一定有問題的，所以在做平行運算之前，要慎重評估是否可行！

在本論文中，我們使用 openMP 與 MPI 進行平行運算，且我們將電腦環境分成個人電腦、國網中心、Amazon EC2，三個地方進行比較，以下是對 openMP 與 MPI 的語法加以介紹。

3.2 OpenMP 平行方法(OpenMP.org)

OpenMP 是用於共享記憶體系統的多執行緒程式設計的一套編譯器指令(OpenMP in Visual C++)。openMP 支持的程式語言包括 C 語言、C++與 Fortran，而支援 openMP 的編譯器包括 Sun Studio、Intel Compiler、gcc4.2.0 以上和 open64 編譯器。

OpenMP 為高階的程式語言，其好處是不用煩惱電腦背後複雜的交錯過程，只要幾個簡單指令即可自動將程式平行化，若忽略 openMP 指令或編譯器不支援 openMP 時，程式可退回原本單執行緒的程式，其指令仍然可以正常運作，只是不能使用多執行緒來加速程式的效率：而共享記憶體指的是在多處理器的計算機系統中，不同的 CPU 使用同一塊記憶體，由圖一看出，多個 CPU 對同一塊記憶體做存取，且 CPU 與記憶體是在同一台機器上運作，因此共享記憶體的架構只能在單一機器上面運作，其優點是存取記憶體時較為容易且沒有傳輸資料的問題，但其缺點是 CPU 的個數較為固定也較難擴充。



圖一. 共享記憶體示意圖

使用 openMP 要先載入 openMP 的標頭檔：omp.h。

$#include<omp.h>

一般使用到 openMP 的指令分成三類：Directives、Clauses、Functions。

$#pragma omp directive [clause]

Directives 部份有 11 個，parallel、sections、for 這三項是拿來執行平行化；master、single、crigica 這三項則是指定使用單一執行緒；atomic、flush、threadprivate 這三項則是用來控制變數的；barrier 是控制執行緒同步用的；ordered 是設定平行化的執行順序，我們用表一說明 Directives 部份的指令用途。

Clauses 的部份有 13 個， copyin、copyprivate、default、shared、private、firstprivate、lastprivate、reduction 這八項是拿來控制變數在平行化時的處理方式；ordered 和 schedule 這兩項是控制平行化執行順序分配方法；if、num_threads 這兩項則是控制執行緒的設定，我們用表二說明 Clauses 部份的指令用途。

Functions 的部份有 22 個，許多函數的使用機會不高，因此只列舉常用的兩個函數，我們用表三說明 Functions 部份的指令用途。

Directives	
parallel	代表接下來的程式將被平行化
for	將接下來的 for 迴圈平行化處理
sections	將接下來的 section 平行化處理
master	指定由主執行緒來執行接下來的程式
single	接下來的程式將只在一個執行緒執行，不被平行化
critical	強制接下來的程式一次只被一個執行緒執行
atomic	記憶體位址將會自動更新
flush	所有執行緒對所有分享的物件有相同的記憶體
threadprivate	對某個執行緒指定變數為不共用的
barrier	同步化，直到所有的執行緒都執行到 barrier
ordered	指定接下來的程式，平行化的 for 迴圈將依序的執行

表一. Directives 部份的指令用途

Clauses	
copyin	使 threadprivate 的變數的值和主執行緒的值相同。
copyprivate	將不同執行緒中的變數共用
default	設定平行化時對變數處理方式的預設值
private	讓每個執行緒中，都有一份變數的複本，以免互相干擾
shared	將變數設定為各執行緒共用
firstprivate	讓每個執行緒中，都有一份變數的複本，以免互相干擾；而起始值則會是開始平行化之前的變數值
lastprivate	讓每個執行緒中，都有一份變數的複本，以免互相干擾；而在所有執行緒都結束後，把最後的值寫回主執行緒
reduction	對各執行緒的變數，指定的運算元來合併寫回主執行緒
ordered	使用於 for，可以在將迴圈平行化的時候，將程式中有標記
schedule	設定 for 迴圈的平行化方法；有 dynamic、guided、runtime、static 四種方法
num_threads	設定平行化時執行緒的數量
if	判斷條件，可以用來決定是否要平行化
nowait	忽略 barrier

表二. Clauses 部份的指令用途

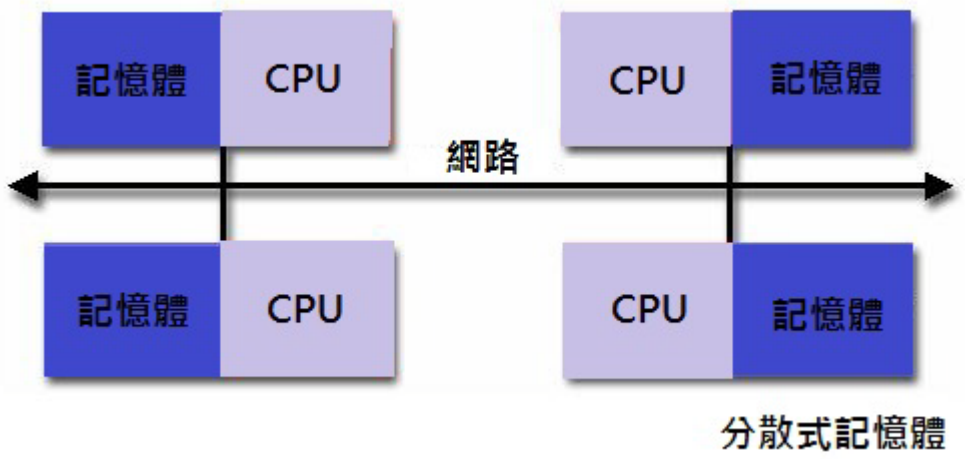
Functions	
omp_get_num_threads	取得目前執行緒的編號
Omp_get_num_procs	取得所有執行緒的個數

表三. Functions 部份的指令用途

3.3 MPI 平行方法(Message passing interface)

MPI 是設計給多電腦系統和異質性網路電腦使用，其介面在基本的傳訊和系統軟體沒有重大改變時，即可在不同介面上運作(Marc Snir and others,1996;Peter S.Pacheco,1997;William Group and others,1999)，MPI 平行程式可以在分散式記憶體平行系統上執行，或在共用記憶體 cluster 平行系統上執行。MPI 支持的程式語言包括 C 語言、C++與 Fortran 等，MPI 的版本有 MPICH、LAMMPI、MVAPICH 等版本。

分散記憶體指的是機器與機器之間透過網路來連結，主要使用在叢集電腦中，透過圖二看出，多台機器透過網路對同一程式做處理，因此可以利用網路串連不同形式的機器，其優點是可以利用網路將全部機器串連，且可以快速的從記憶體存取，但其缺點是工程師須花更多時間在 CPU 與 CPU 之間的資料交換，還有網路速度快慢的問題！



圖二. 分散式記憶體示意圖

使用 MPI 要先載入 MPI 的標頭檔：mpi.h。

$#include<mpi.h>

使用 MPI 要先設定四個基本函數且在所有程式中只呼叫一次，我們用表四來說明四個基本函數的用途。

參與平行計算的各個 CPU 之間資料傳遞方式有兩種，一種叫做〝點對點通訊〞，其意思為一個 CPU 與另一個 CPU 之間的資料傳送；另一種叫做〝集體通訊〞，其意思是每一個 CPU 都要參與運作。

我們使用表五說明點對點通訊的函數用途，特別的是一個 MPI_Send 必須要一個對應的 MPI_Recv 配合，且 MPI_SendRecv 作用等於一個 MPI_Send 加一個 MPI_Recv。

我們使用表六說明集體通訊的函數用途，與點對點通訊不同的是每一個 CPU 都必須呼叫同一個函數才能完成資料的傳送。

平行程式的切割方式分成兩種，一種是計算切割而資料不切割，其缺點是不能夠節省記憶體的使用量，優點是程式容易閱讀且容易維修，另一種是計算和資料都切割，其缺點是程式閱讀與維修較困難，優點是能夠節省記憶體的使用量；平行程式的方式大約分成四種，一種是無邊界資料交換程式，其意思是每個 CPU 都單獨計算結果後，回收到 master CPU 上，slave CPU 互相沒有關聯，一種是需要邊界資料交換程式，其意思是當某個 CPU 在計算時，需要另外一個 CPU 計算的值，則 CPU 之間就必須要交換資料，一種是維度不能整除程式，其意思是若維度為 23，而 CPU 個數有 4 個，造成分割計算或資料不等長，造成資料傳遞上的不一致，最後一種是多維陣列程式。

我們使用表七說明其他常使用函數的用途。在本論文中，主要使用計算切割但資料不切割且無邊界資料交換的模式攥寫程式。

基本指令	
MPI_Init()	啟動該程式在多個 CPU 上的平行計算工作
MPI_Finalize()	結束平行計算工作
MPI_Comm_size()	得知參與平行計算的 CPU 個數
MPI_Comm_rank()	得知是第幾個 CPU

表四.四個基本函數的用途

點對點通訊指令	
MPI_Send()	送出資料的 CPU 叫用
MPI_Recv()	接收資料的 CPU 叫用
MPI_Sendrecv()	接收 A CPU 資料又要傳送給 B CPU 資料時叫用

表五.點對點通訊函數的用途

集體通訊指令	
MPI_Scatter()	將資料切割並傳送計算
MPI_Gather()	將計算收集並合併結果
MPI_Allgather()	把運算結果存放再每一個 CPU
MPI_Reduce()	運算結果存放在指定的 CPU
MPI_Allreduce()	運算結果存放在每一個 CPU
MPI_Bcast()	將同一項資料傳送給各個 CPU
MPI_Scatterv()	類似 MPI_Scatter，可分送不等長資料
MPI_Gatherv()	類似 MPI_Gather，可收集不等長資料

表六.集體通訊函數的用途

其他指令	
MPI_Pack()	集結不同或相同的資料型態
MPI_Unpack()	分解不同或相同的資料型態
MPI_Barrier()	將所有 CPU 達到同步的狀況
MPI_Wtime()	取得平行程式執行時的時鐘時刻

表七.其他常用函數的用途

4. 國家高速網路與計算中心與雲端運算 Amazon EC2 環境介紹

4.1 國家高速網路與計算中心(國研院國網中心)

國家高速網路與計算中心現有的平行計算環境有 IBM Cluster 1350、IBM P690、HP Superdome、HP XC6000 Cluster 以及 Formosa II HPC Cluster。本文所使用的叢集環境為 Formosa II HPC Cluster，圖三顯示所有電腦的規格與配備。

Formosa II 是採用 PBS Torque 工作排程軟體，使用者必須備妥 PBS Torque 的 job command file，使用 qsub 指令把該批次工作交給 Formosa II 來執行，且 Formosa II 登入主機的方式是使用 putty 軟體，而傳輸檔案是以 WinSCP 軟體。

本論文所使用到的軟體是 C 語言與 R 統計軟體，編譯 C 語言要在 Command line 下執行，接著存入 R 的命令稿，接著用 qsub 指令把 job script 裡的執行檔交給 Formosa II 執行，我們用以下的範例來說明執　行程式的過程與 job script 的環境設定。

型號	IBM 1350	IBM 1350A	Formosa 2	HP LC6000	IBM P595	IBM P690	IBM P690A	HP Superdome	HP Superdome 2
架構	叢集系統	叢集系統	叢集系統	叢集系統	SMP	SMP	SMP	SMP	SMP
硬體規格	Intel 3.0 GHz Dual-Core	Intel 3.0 GHz Quad-Core	AMD Opteron 275 Dual Core	Intel Itanium 2 1.5 GHz	IBM Power 5 1.9GHz	IBM Power 4 1.3 GHz	IBM Power 4 1.9 GHz	Intel 1.5 GHz	Intel 1.6 GHz Dual Core
記憶體(Per Node)	16 GB	16 GB	8 GB	8 GB	256 GB	128 GB	64 GB	128 GB	1 TB
Node	512	128	80	192	1	8	3	2	1
CPU	1024	256	160	384	64	256	96	128	44
Core	2048	1024	320	384	64	256	96	128	44
Core/Node	4	8	4	4	64	32	32	64	88
網路連接	Infiniband	Gigabit Ethernet	Infiniband	Infiniband Quadric Elan 4	--	--	--	--	--
計算效能 (Tflops)	約20	約7	約1	約2	0.4	0.7	0.4	0.6	0.5
作業系統	SuSE	SuSE	Fedora Linux	RedHat AS 4.5 Compatible	AIX	AIX	AIX	HP-UX	HP-UX
排程軟體	LoadLeveler	LoadLeveler	PBS Torque	LSF	LoadLeveler	LoadLeveler	LoadLeveler	LSF	LSF

圖三. 國家高速網路與計算中心的電腦規格(資料來源：國家高速網路與計算中心)

4.1.1 一般 k-means 分群方法程式流程

1.$gcc –fPIC –c kmean1.c –o kmean1.o

2.$gcc –fPIC –c kmean2.c –o kmean2.o

3.$ld –r –o KMEAN.o kmean1.o kmean2.o

4.$gcc –shared KMEAN.so KMEAN.o

5.$R --save < kmeans.R

6.$qsub job.sh(執行 R --save < kmeans exe > kmeans.out&)

第 1 和 2 步驟是將 c 檔編譯成 o 檔的指令，第 3 和 4 步驟是將 o 檔包裝成一個 so 檔案的 library 給 R 呼叫，第 5 步驟是把讀取寫好 R 的命令稿，接著第 6 步驟則是執行 R 的執行檔，qsub job.sh 只能使用在國網中心上，而個人電腦以及 Amazon EC2 則是使用括號中的執行指令。

我們使用圖四呈現 job.sh 的內容，在圖中第一段的第 2 行代表的是這個程式的名字，第 6 行表示的是這程式用到 1 個 CPU，第 2 段為國網中心的基本設定，第 3 段為執行 R 的執行檔指令。

4.1.2 使用 OpenMP 平行運算的流程

1.$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gcc/4.3.2/
lib64:/opt/mpfr_2.3.1/lib:/opt/gmp_4.2.2/lib/

2.$gcc –fPIC –fopenmp –lgmp –c kmean1.c –o kmean1.o

3.$gcc –fPIC –fopenmp –lgmp –c kmean2.c –o kmean2.o

4.$ld –r –o KMEAN.o kmean1.o kmean2.o

5.$gcc –shared KMEAN.so KMEAN.o –fopenmp –lgmp

6.$R --save < kmeans.R

7.$qsub job_omp.sh(執行 R --save < kmeans exe > kmeans _omp.out&)

第 1 步驟是設定 gcc 的 library 版本，因為在國網中心的 gcc 版本有相當多種，因此要設定 gcc4.2 以上的版本才能順利執行 OpenMP，在個人電腦以及 Amazon EC2 上則不用使用此指令，第 2 和 3 步驟是將 c 檔編譯成 o 檔的指令，其中加入-fopenMP 與-lgmp 才能順利使用 OpenMP 的指令，第 4 和 5 步驟是將 o 檔、-fopenMP 以及-lgmp 包裝成一個 so 檔案的 library 給 R 呼叫，第 5 步驟是把讀取寫好 R 的命令稿，接著第 6 步驟則是執行 R

的執行檔，qsub job_omp.sh 只能使用在國網中心上，而個人電腦以及 Amazon EC2 則是使用括號中的執行指令。

我們使用圖五呈現 job_omp.sh 的內容，在圖中第一段的第 2 行代表的是這個程式的名字，第 6 行表示的是這程式用到 4 個 CPU，第 2 段為國網中心的基本設定以及設定 gcc 的 library 版本，第 3 段為執行 R 的執行檔指令。

4.1.3 使用 MPI 平行運算的流程

1.$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mpich/gnu /lib

2.$/opt/mpich/gnu/bin/mpicc –fPIC –c kmean1.c –o kmean1.o

3.$/opt/mpich/gnu/bin/mpicc –fPIC –c kmean2.c –o kmean2.o

4.$ld –r –o KMEAN.o kmean1.o kmean2.o

5.$/opt/mpich/gnu/bin/mpicc –shared KMEAN.so KMEAN.o

6.$R --save < kmeans.R

7.$qsub job_mpi.sh(執行 R --save < kmeans exe > kmeans _mpi.out&)

第 1 步驟是設定 mpi 的 gcc library 版本，在個人電腦以及 Amazon EC2 上則不用使用此指令，第 2 和 3 步驟是將 c 檔編譯成 o 檔的指令，其中要使用 mpicc 的編譯器才能順利使用 MPI 的指令，第 4 和 5 步驟是將 o 檔包裝成一個 so 檔案的 library 給 R 呼叫，第 5 步驟是把讀取寫好 R 的命令稿，接著第 6 步驟則是執行 R 的執行檔，qsub job_mpi.sh 只能使用在國網中心上，而個人電腦以及 Amazon EC2 則是使用括號中的執行指令。

我們使用圖六呈現 job_mpi.sh 的內容，在圖中第一段的第 2 行代表的是這個程式的名字，第 6 行表示的是這程式用到 16 個 CPU，第 2 段為國網中心的基本設定以及設定 mpi 的 gcc library 版本，第 3 段為執行 R 的執行檔指令。

```
### Job Name
#PBS -N SERIAL_JOB
### Submits the job to the serial queue
#PBS -q serial
### Requests 1 nodes, 1 CPU.
#PBS -l nodes=1:ppn=1
### Output files
#PBS -o pbs.log
#PBS -e pbs.err
### Declare job non-rerunable
#PBS -r n

# Change to working directory
cd $PBS_O_WORKDIR
# Setting environment variable
P4_GLOBMEMSIZE=99187896
export P4_GLOBMEMSIZE

echo "Starting on `hostname` at `date`"
###8. To run the program in the batch file
/opt/R-2.9.2/bin/R CMD BATCH /home/username/kmean/kmeansexe kmeans_omp.out
echo -e "Hello! \a \n"
echo "Job Ended at `date`"
```

圖四. job.sh 的環境設定與內容.

```
### Job Name
#PBS -N PARALLEL_JOB
### submits the job to the default queue
#PBS -q parallel
### requests 1 nodes, and each node has 4 CPUs.
#PBS -l nodes=1:ppn=4
### Output files
#PBS -o pbs.log
#PBS -e pbs.err
### Declare job non-rerunable
#PBS -r n

### Change to working directory
cd $PBS_O_WORKDIR
###Setting environment variable
P4_GLOBMEMSIZE=99187896
OMP_NUM_THREADS=4
export P4_GLOBMEMSIZE
export OMP_NUM_THREADS
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gcc/4.3.2/lib64
        :/opt/mpfr_2.3.1/lib:/opt/gmp_4.2.2/lib/

echo "Starting on `hostname` at `date`"
###8. To run the program in the batch file
/opt/R-2.9.2/bin/R CMD BATCH /home/username/kmean/kmeansexe kmeans_omp.out
echo -e "Hello! \a \n"
echo“Job Ended at `date`"
```

圖五. job_omp.sh 的 openMP 環境設定與內容

```
### Job Name
#PBS -N PARALLEL_JOB
### Submits the job to the n4 queue
#PBS -q n4
### Requests 4 nodes, and each node has 4 CPUs. Total 16 CPUs.
#PBS -l nodes=4:ppn=4
### Output files
#PBS -o pbs.log
#PBS -e pbs.err
### Declare job non-rerunable
#PBS -r n

# Change to working directory
cd $PBS_O_WORKDIR
# Setting environment variable
P4_GLOBMEMSIZE=99187896
export P4_GLOBMEMSIZE
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mpich/gnu/lib

echo "Starting on `hostname` at `date`"
###8. To run the program in the batch file
/opt/R-2.9.2/bin/R CMD BATCH /home/username/kmean/kmeansexe kmeans_mpi.out
echo -e "Hello! \a \n"
echo "Job Ended at `date`"
```

圖六.job_mpi.sh 的 MPI 環境設定與內容

4.2 雲端運算

雲端運算是一種基於網際網路的新運算方式，透過網際網路上異構、自治的服務為個人和企業使用者提供按需即取的運算。因為資源是在網際網路上，而在電腦流程圖中，網際網路常以一個雲狀圖案來表示，因此稱為雲端。

雲端運算的資源是動態易擴充且虛擬化的，透過網際網路提供，終端使用者不需要了解雲端中基礎設施的細節，也不必具有相應的專業知識，只要關注在自己真正需要什麼樣的資源與如何透過網路得到相應的服務。

目前雲端運算的應用有許多種，我們將使用 Amazon 的 EC2 開創需要的硬體設備，以下是對 Amazon EC2 的使用加以介紹！

4.2.1 Amazon EC2 介紹(Amazon web services)

Amazon EC2(Elastic Compute Cloud) 為 Amazon 所開發的雲端運算服務環境，是一個提供客戶租賃虛擬的執行環境，以供企業開發、測試或執行自己的應用程式，客戶可以選擇每個執行環境的規格，涵蓋記憶體空間、運算單位及儲存空間等。

1. 登入 Amazon EC2 步驟如下：

(1)在 Amazon web services 上申請帳號並輸入信用卡號

(2)在 Account 選擇 Security Credentials->Access Keys

按下 Create a new Access key，且紀錄 Access key ID 與 Secret Access key

(3)安裝 Elasticfox 附加元件在 Firebox 上

(4)開啟 Firebox->選擇工具->Elasticbox(登入管理機器的介面)

(5)在 Manage EC2 Credentials 視窗, 輸入"Account Name", "AWS Access Key", "AWS Secret Access Key“

4.2.2 登入 Amazon EC2 的事前工作(Amazon EC2 使用操作筆記)

登入 Amazon EC2 且設定好登入資訊，以後就都可以從 Elasticfox 上管理 EC2 的機器，接下來還有一些事前工作需要設定好，才完成開啟 EC2 機器的動作。

1. Key Pairs(登入 EC2 機器的密碼)：

(1)在 Elasticfox 上，選擇"KeyPairs"，按下 Create a new keypair，輸入 name，產生 name.pem 檔下載視窗，將此下載儲存。

(2)下載 puttygen 開啟後，選擇"Conversion"中的 Import key，載入 name.pem，按下 Save private key，儲存轉檔好的.ppk 檔。

2. Security Group(防火牆的設定)：

(1)在 Elasticfox 上，選擇"Security Groups"，左手邊按下 Create Security Group，輸入 group name。

(2)選擇左手邊的 group name，右手邊按下 Grant Permission，設定要防火牆的初始值，並且在 Host/Networks Details 點選 Get My Host Address 後，按下 add，即設定完成。

3. AMI(EC2 虛擬機器的映像檔)：

(1)在 Amazon web services 首頁上，選擇 Amazon EC2，在左下角選擇 Amazon Machine Images(AMIs)。

(2)由上面挑選映像檔，找到 ami-78b15411(Fedora core 6) 、ami-c0f615a9(Ubuntu 8.10)，這兩個的機器大小有 m1.small 與 c1.medium，ami-1b9b7c72(安裝好 R 與 gcc 的 Ubuntu 8.10)，這個的機器大小有 m1.large、m1.xlarge、m2.xlarge、m2.2xlarge、m2.4xlarge、c1.xlarge 這六種，這三個映像檔會在接下來加以詳述，其他的映像檔在 Amazonweb services 網頁上有詳細介紹。而越大的記憶體或 CPU，其價格也會有所差別，所以選擇好需要的記憶體、CPU 與作業系統版本，就可以執行開啟機器的動作了！

(3)表八詳細陳述 Amaonz EC2 的機器大小與規格。

4.2.3 連結 Amazon EC2 機器

1. 創建 EC2 機器：

(1)在 Elasticfox 上，選擇"Images"，輸入之前選擇的 AMI ID，點選映像檔後，按下 Launch instance 或按右鍵點 Launch instance of this AMI，設定好所有資訊，點選 Launch 即可！

類型	記憶體 (GB)	CPU	ECU	硬碟記憶體 (GB)	平台	Linux 價格 (USD/HR)
m1.small	1.7	1	1	160	32	0.085
m1.large	7.5	2	4	850	64	0.34
m1.xlarge	15	4	8	1690	64	0.68
m2.xlarge	17.1	2	6.5	420	64	0.5
m2.2xlarge	34.2	4	13	850	64	1.2
m2.4xlarge	68.4	8	26	1690	64	2.4
c1.medium	1.7	2	5	350	32	0.17
c1.xlarge	7	8	20	1690	64	0.68

表八. Amazon EC2 的規格與價格

(2)在 Elasticfox 上選擇"Instances"，就會看到新 run 的機器（Status 是 running 才算完成）。

2. 設定 Elastic IP(登入 EC2 機器的 IP 位址)：

(1)在 Elasticfox 上，選擇"Elastic IPs"，按下 Allocate a new address，自動產生一個新 IP。

(2)在 Elasticfox 上，選擇"Instances"，點選要加載固定 IP 的機器，按右鍵點 Associate Elastic IP with Instancec 後，再點選 Associate，即為加載完成!

3. 連線 EC2 機器：

本論文連線 EC2 機器所使用的軟體為 putty 軟體，而傳輸檔案到 EC2 機器所使用的傳輸工具為 WinSCP。

(1)在 Elasticfox 上，選擇"Instances"，按右鍵點 View Details，找到 Public DNS Name (ex:ec2-....compute-1.amazonaws.com)。

(2)在 putty 上的 Host name 輸入 Public DNS Name 或者加載好的 Elastic IP

(3)在 putty 上左邊選單點選 Connection->SSH->Auth，在 Private key file for authentication 選擇 key pair 的.ppk 檔。

(4)按下 open，輸入 root，即為登入完成。

4. 關掉 EC2 機器：

(1)在 Elasticfox 上，選擇"Instances"，點選要關機的機器，按下 Terminate Selected Instances，即為關機完成。

(2)也可在登入機器後，打入 halt 指令，極為直接關機。

4.2.4 安裝軟體

1. 安裝 gcc compiler：

(1)在 Fedora core 6 機器上輸入

$yum check-update

$yum install gcc

(2)在 Ubuntu 8.10 機器上輸入

#apt-get update

#apt-get install gcc

註：在 Fedora core 6 機器上出現錯誤(Error: Cannot find a valid baseurl for repo: extras)時，更改路徑”/ect/yum.repos.d/”下的.repo 檔，將裡面所有的

#baseurl=http://download.fedora.redhat.com/pub/fedora/linux/core/$releasever/$basearch/os/

改成

baseurl=http://archives.fedoraproject.org/pub/archive/fedora/linux/core/$releasever/$basearch/os/

2. 安裝 R(The R Project for Statistical Computing)：

(1)安裝 perl 軟體

#yum(apt-get) install perl

(2)解 R 的壓縮檔

#tar zxvf R-2.10.1.tar.gz

(3)安裝 R

#./configure #make #make install

註：若沒有 realline library 或 X11 library，則要先安裝或者可以打

#./configure --with-readline=no --with-x=no

4.2.5 架設 MPI 叢集環境(Official Python Planet@Taiwan)

我們希望在 Amazon EC2 上架設 MPI PC cluster，其架設的步驟如下：

1. 更改/etc/hosts，方便遠端直接快速連結

假設有兩個MPI node，所有的node的/etc/hosts皆需要修改

master@mpi-1:~$ sudo vi /etc/hosts

master@mpi-2:~$ sudo vi /etc/hosts

修改內容如下:

IP mpi-1

IP mpi-2

2. 安裝與設定 NFS，讓所有 node 有共享的 master folder

我們以 mpi-1 做為 master

master@mpi-1:~$ sudo apt-get update

master@mpi-1:~$ sudo apt-get install nfs-kernel-server

設定 NFS 目錄與掛載權限

master@mpi-1:~$ sudo mkdir /mpi

master@mpi-1:~$ sudo vi /etc/exports

修改內容如下:

/mpi*(rw,sync)

master@mpi-1:~$ sudo chown master:master /mpi

master@mpi-1:~$ sudo /etc/init.d/nfs-kernel-server restart

在其他的 node 掛載 mpi-1 master folder

master@mpi-2:~$ sudo mkdir /mpi

master@mpi-2:~$ sudo chown master:master /mpi

master@mpi-2:~$ sudo apt-get install nfs-common

master@mpi-2:~$ sudo mount -t nfs mpi-1:/mpi /mpi

3.安裝 ssh，設定金鑰讓各 node 之間以 master 帳號連結不用輸入密碼

主要是為了方便往後可在 scripts 裡設定各 node 自動串連

master@mpi-1:~$ sudo apt-get install ssh

master@mpi-2:~$ sudo apt-get install ssh

master@mpi-1:~$ ssh-keygen -t dsa

master@mpi-1:~$ scp .ssh/id_dsa.pub master@mpi-2:~/mpi-1.pub

master@mpi-2:~$ cat mpi-1.pub >> .ssh/authorized_keys

master@mpi-2:~$ ssh-keygen -t dsa

master@mpi-2:~$ scp .ssh/id_dsa.pub master@mpi-1:~/mpi-2.pub

master@mpi-1:~$ cat mpi-2.pub >> .ssh/authorized_keys

測試登入（不用輸入密碼）

master@mpi-1:~$ ssh mpi-2

master@mpi-2:~$ ssh mpi-1

4.安裝 MPICH2

只要在 mpi-1 master folder 安裝一次即可

下載 MPICH2 的網址如下:

http://www.mcs.anl.gov/research/projects/mpich2/

下載完畢後放入 mpi-1 的/mpi 目錄夾中

master@mpi-1:/mpi $ sudo apt-get install build-essential

master@mpi-1:/mpi $ mkidr mpich2

master@mpi-1:/mpi $ tar xzvf mpich2-1.1.1p1.tar.gz

master@mpi-1:/mpi $ cd mpich2-1.1.1p1

master@mpi-1:/mpi/mpich2-1.1.1p1 $ ./configure --
prefix=/mpi/mpich2

master@mpi-1:/mpi/mpich2-1.1.1p1 $ make

master@mpi-1:/mpi/mpich2-1.1.1p1 $ sudo make install

設定 PATH 環境變數

master@mpi-1:~$ vi .bashrc

master@mpi-2:~$ vi .bashrc

export PATH="/mpi/mpich2/bin:$PATH"

export LD_LIBRARY_PATH=
"/mpi/mpich2/lib:$LD_LIBRARY_PATH"

在所有 node 上測試程式路徑是否正確

master@mpi-1:~$ which mpd

master@mpi-1:~$ which mpiexec

master@mpi-1:~$ which mpirun

master@mpi-2:~$ which mpd

master@mpi-2:~$ which mpiexec

master@mpi-2:~$ which mpirun

5.設定 MPICH2

在所有 node 上建立 mpd.conf 及 mpd.hosts

master@mpi-1:~$ vi mpd.hosts

master@mpi-2:~$ vi mpd.hosts

修改內容如下:

mpi-1

mpi-2

master@mpi-1:~$ echo MPD_SECRETWORD=xxx-xx > .mpd.conf

master@mpi-1:~$ chmod 600 .mpd.conf

master@mpi-2:~$ echo MPD_SECRETWORD=xxx-xx > .mpd.conf

master@mpi-2:~$ chmod 600 .mpd.conf

啟動所有 node 的 mpi service

master@mpi-1:~$ mpd &

master@mpi-1:~$ mpdboot -n 2

master@mpi-2:~$ mpd &

master@mpi-2:~$ mpdboot -n 2

在所有 node 上檢查是否運行成功

master@mpi-1:~$ mpdtrace

mpi-1

mpi-2

master@mpi-2:~$ mpdtrace

mpi-1

mpi-2

6. 安裝完畢！

5. 潛在群體模型

令 $Y_i=(Y_{i1},Y_{i2},\cdots,Y_{iM})^T$ 為 M 個觀測值，且令 S_i 為為 N 個樣本中第 i 個所屬的潛在群體且 $S_i\in\{1,\cdots,J\}$。

潛在群體模型架構在條件獨立的概念，其意思是不管在哪個族群，觀測值皆會互相獨立，因此密度函數可表示成

$$f(y_1,\cdots,y_M)=\sum_{j=1}^{J}\{\eta_j\prod_{m=1}^{M}f_j(y_m)\}$$

$$\text{其中 } \Pr(S_i=j)=\eta_j \text{ 且 } Y_{im}\mid S_i=j\sim f_j(\cdot)$$

許多作者會將模型延伸到可加入與潛在群體有關的風險因子或者在潛在群體影響觀測值的變數。

令 $x_i=(1,x_{i1},x_{i2},\cdots,x_{ip})^T$ 變數為與潛在群體有關的風險因子，我們使用廣義線性模型(generalized linear models)去聯結 S_i 與 x_i 之間的影響

$$\log[\frac{\eta_j(x_i)}{\eta_J(x_i)}]=\beta_{0j}+\beta_{1j}x_{i1}+\cdots+\beta_p x_{jip}\text{，其中 } j=1\cdots J-1$$

令 $z_i=(z_{i1},\cdots,z_{iM})$ 為在潛在群體影響觀測值的變數，其中 $z_{im}=(z_{im1},\cdots,z_{imL})^T\ m=1\cdots M$，也就是 $f_j(\cdot)=f_j(\cdot\mid z_{im})$。

當觀測值為連續時，我們令 $(Y_{im}\mid S_i=j,z_{im})\sim N(\mu_{mj}(z_{im}),\sigma_m^2)$，且 $\mu_{mj}(z_{im})=\theta_{mj}+\tau_{1m}z_{im1}+\cdots\tau_{Lm}z_{imL}$。

因此，加入調整變數後，條件獨立假設成為 $f_j(y_1,\cdots,y_M\mid z_i)=\prod_{m=1}^{M}f_j(y_m\mid z_{im})$。

Huang, Wang & Hsu (manuscript) 提出利用分群分析對潛在群體模型做參數估計。本論文引用了群聚方法中兩種不同的 k-means 分群方法，將原本的距離測度改成相關係數或共變異數，然後對所有的主體分群，使得屬於在同一群的主體所測得的各項目能相互獨立，再將估計出的潛在群體 S_i 視為已知變數後，在估計潛在群體迴歸分析模型的參數即可。

我們先介紹如何將距離測度改成相關係數或共變異數，第一，定義觀測值之間的樣本共變異數矩陣，對連續的觀測值，組成樣本共變異數矩陣的成份為 Y_{im} 與 Y_{it} 的樣本共變異數。

第二，我們令 $ACov_j$ 為第 j 群之樣本共變異數矩陣非對角線元素（對於連續）/非對角區塊（對於離散）之絕對值平均，且令

$\mathrm{LoI}=\sum_{j=1}^{J} w_j ACov_j$ ，$w_j=\frac{\text{第}j\text{群體的資料個數}}{N}$，其值越小代表越趨近獨立。

接著，我們將介紹如何用兩種 k-means 演算法去估計潛在群體。

Non-updated k-means 分群方法的演算法：

nk1.隨機將資料分成 k 群

nk2.使用初始群體將每個資料變動到各群計算 $\mathrm{LoI}_i^{(u)}$ 值且將資料分配具有最小 $\mathrm{LoI}_i^{(u)}$ 值的群體中，當所有資料分配完成後才將初始群體用新的分群結果替代

nk3.反覆執行 nk2 直到全部資料皆沒有移動為止

Updated k-means 分群方法的演算法：

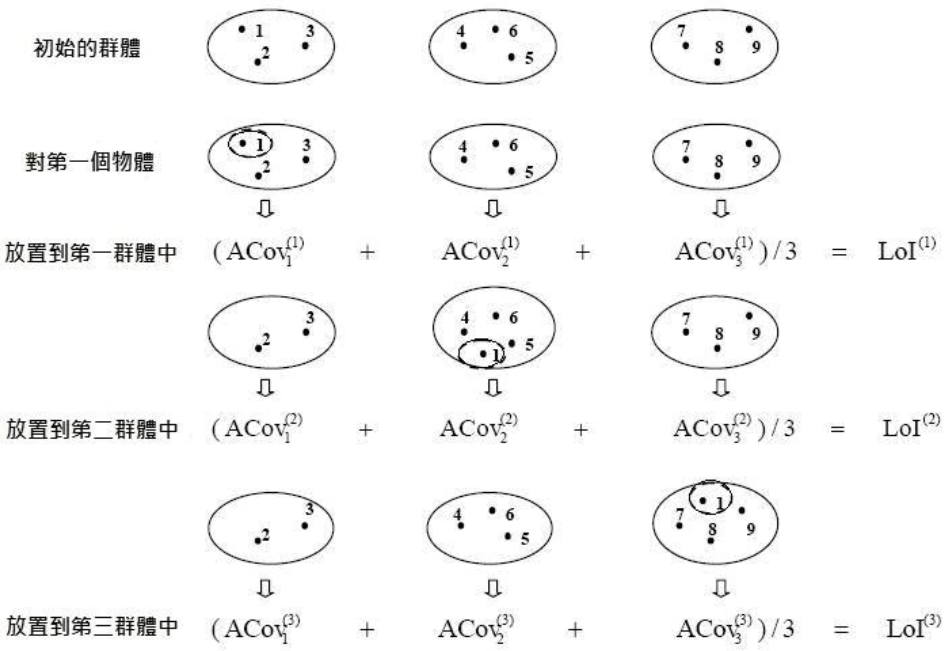
uk1.隨機將資料分成 k 群

uk2.使用初始群體將每個資料變動到各群計算 $\mathrm{LoI}_i^{(u)}$ 值且將資料分配具有最小 $\mathrm{LoI}_i^{(u)}$ 值的群體中，當有資料被移動時，初始群體也隨之改變

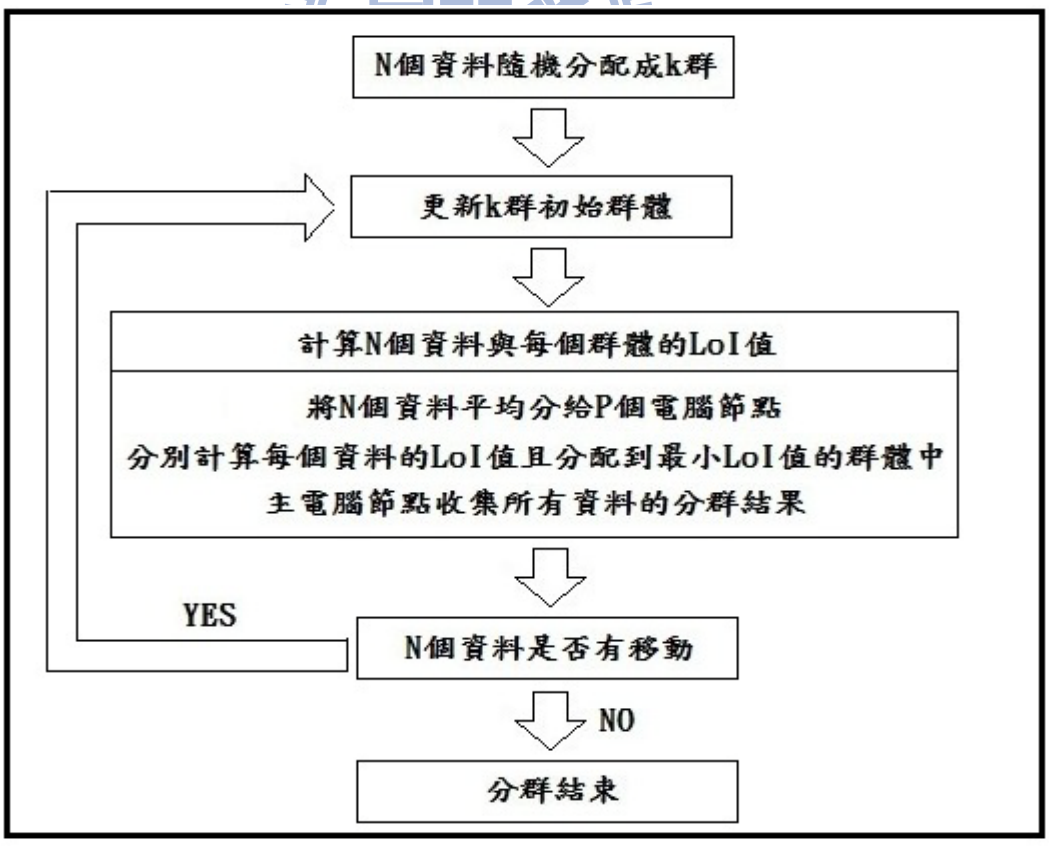
uk3.反覆執行 uk2 直到全部資料皆沒有移動為止

註：在 nk2 與 uk2 中，我們定義 $\mathrm{LoI}_i^{(u)}=\sum_{j=1}^{J} w_j ACov_j$ ，$w_j=\frac{\text{第}j\text{群體的資料個數}}{N}$，且 $\mathrm{LoI}_i^{(u)}$ 為第 i 個資料被變動到第 u 個族群的值，將其資料變動到 J 個族群後得到 $LoI_i^{(1)},\cdots,LoI_i^{(J)}$，選出最小的值 $\mathrm{LoI}_i^{(u)}$，則代表第 i 個資料被分配到第 u 個族群裡，我們使用圖七說明 $\mathrm{LoI}_i^{(u)}$ 值的計算過程。

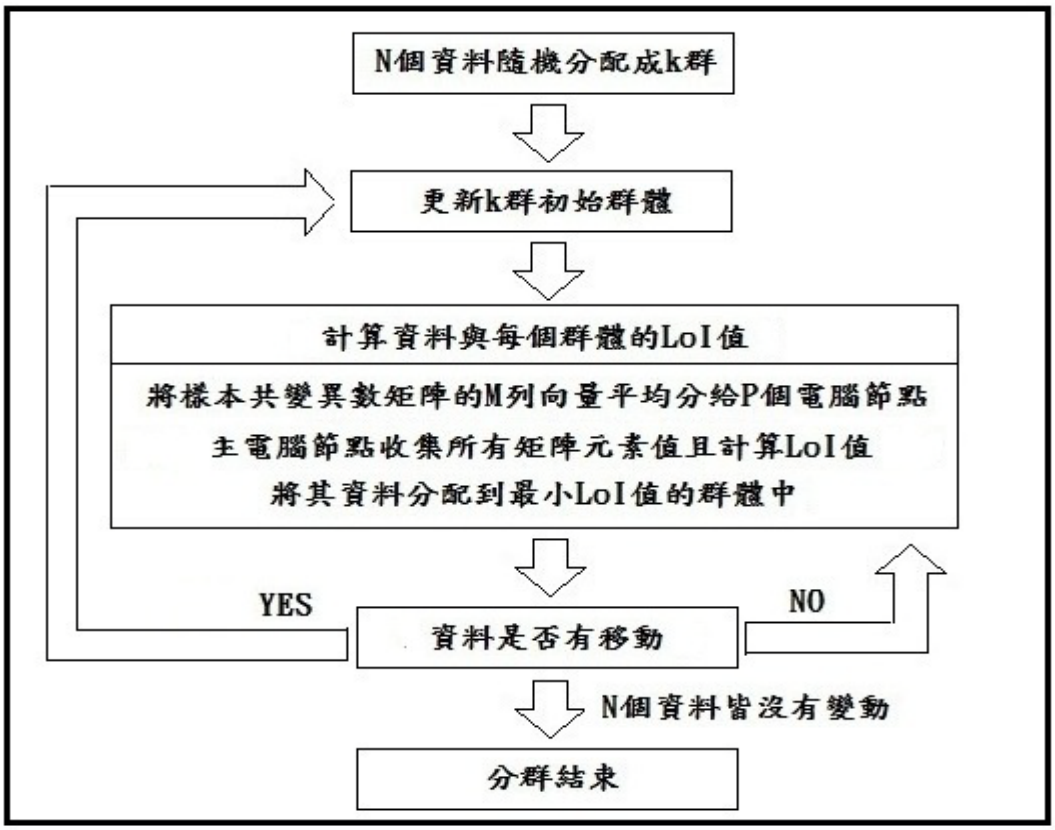
本論文主要目的是對 k-means 分群方法估計潛在群體的計算過程做平行運算，在 Non-updated k-means 分群方法中，我們將 N 個資料平均分配給 P 個電腦節點，分別計算每個資料的分群結果，在主電腦節點上收集所有資料的分群結果替代初始群體，在 Updated k-means 分群方法中，一有資料變動，則初始群體也隨之變動，在平行運算上具有相依性，會造成計算上的錯誤，因此我們對計算 $\mathrm{LoI}_i^{(u)}$ 值的樣本共變異數矩陣做平行運算，將 M 列向量平均分給 P 個電腦節點，分別計算其矩陣的元素，在主電腦節點上收集矩陣元素值且計算 $\mathrm{LoI}_i^{(u)}$ 值，我們使用圖八與圖九說明兩種分群方法的平行流程。



圖七. $LoI_i^{(u)}$ 值的計算過程



圖八. Non-updated k-means 分群演算法平行過程



圖九. Updated k-means 分群演算法平行過程

6. 實驗結果

6.1 乳癌資料

乳癌資料來自於使用基因表現量預測乳癌結果的研究（van't Veer and other,2002）。78 名年齡在 55 歲左右且有乳癌預兆基因的陣發性陰性淋巴結乳癌病人，其中 44 名病人在 5 年間沒有復發病徵兆（預測良好組，追蹤時間的平均值為 8.7 年），剩下 34 名病人在 5 年間有轉移擴散的現象（預測不良組，轉移現象發生時間的平均值為 2.5 年）。此資料記載了紅色與綠色螢光基因表現量的平均值，其紅色代表誘發程度且綠色代表抑制程度，而 p-value 代表基因表現量的差異，其值越小代表差異越大。

對於基因表現量分析的演算法都需要完整的基因晶片矩陣。例如:k-means 分群方法無法有效處理包含遺漏值的資料，因此，有必要在不完整的資料中填補遺漏值，以降低不完整資料分析的有效性與增加資料的範圍。K-nearest neighbors(KNN)方法可以有效且精準的估計遺漏值，KNN 方法所選擇的基準基因與包含遺漏值的基因相似，例如:在試驗 1 中基因 A 有遺漏值，我們使用 KNN 方法選出在試驗 2 到 N 中最相似的 K 個基因，其中 N 為試驗的總個數。使用其 K 個在試驗 1 中之基因表現量的加權平均值去估計在試驗 1 中基因 A 的遺漏值，其中權重為每個基因與基因 A 的相似程度，越相似其權重越高。

接著透過兩個步驟去篩選基因，第一步驟從 24481 個基因選取 4741 基因，其來自於基因表現量比大於 2 或小於 0.5（差距超過兩倍），且觀測基因表現量差異的 P-value 小於 0.01，其中要超過 3 個樣本擁有這種基因型，其主要用意是選出最有差異性的基因，第二步驟，我們使用組間差異平方和與組內差異平方和的比率來選取基因（Dudoit and other,2002），其比率表示為

$$BW(m)=\frac{\sum_i\sum_c I(d_i=c)(\bar{y}_{cm}-\bar{y}_{\cdot m})^2}{\sum_i\sum_c I(d_i=c)(y_{im}-\bar{y}_{cm})^2}$$

其中 y_{im} 代表第 i 個樣本第 m 個基因表現量的比率，d_i 為第 i 個樣本屬於預測良好或預測不良的指標，$\bar{y}_{cm}$ 為屬於 c 預測群的第 m 個基因表現量之平均，$\bar{y}_{\cdot m}$ 為所有樣本的第 m 個基因表現量之平均，我們使用上述式子計算每個基因的 BW 比率且選出 BW 比率較大的基因作為模型所使用的資料。

6.2 分群結果與時間效率

6.2.1 OpenMP 運算結果

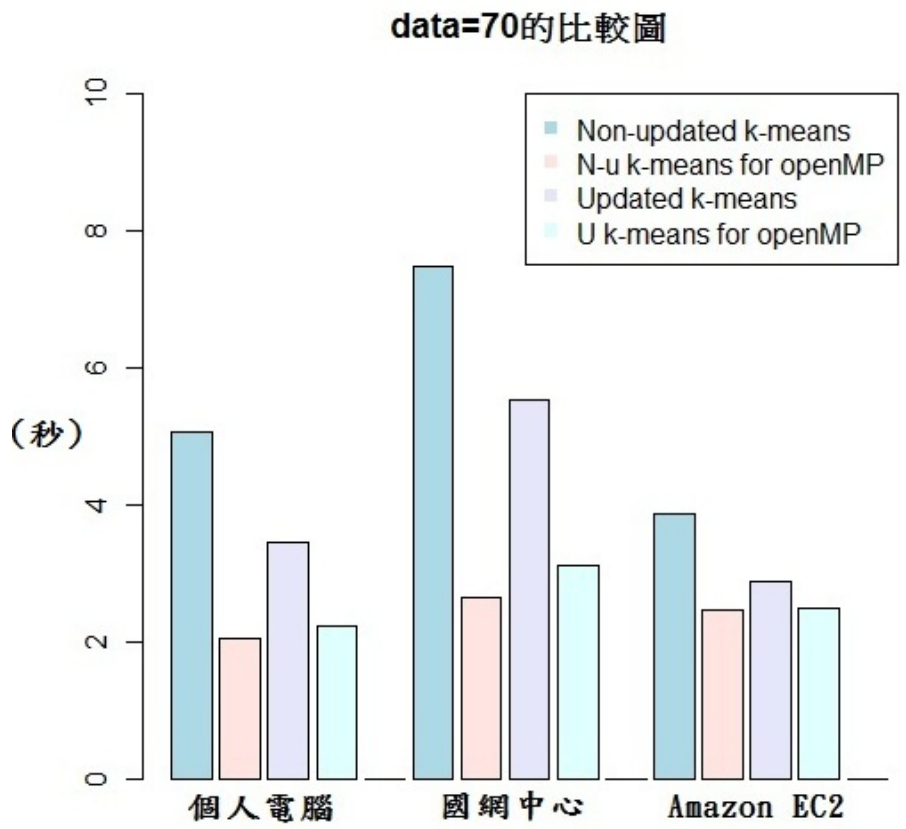
我們從 4741 個基因選出 BW 比率較大的 70~500 個基因作為分群的背景資料，對三個不同的電腦環境執行程式，比較兩種不同 k-means 分群方法平行與否的差異，圖十的資料是使用 BW 比率最大的 70 個基因做為觀測值，我們可以看到 Non-updated k-means 分群方法的遞迴次數較 Updated k-means 分群方法多，因此程式執行時間也較久，但兩個分群方法所分群的結果皆為一致，且使用 openMP 平行運算的程式都有明顯的減少執行時間，圖十一~圖十五使用 BW 比率最大的 100~500 個基因，我們也發現觀測值越多，使用 openMP 平行運算的效率愈好，因此可執行於往後的較大量的資料，詳細的運行時間與分群結果收錄在附錄一到附錄六中。

接著我們發現同一程式在 Amazon EC2 的運行時間較快，而平行過後的程式在個人電腦的運行時間較快，我們在表九附上了三種不同電腦環境的硬體規格，以了解其之間的差異。

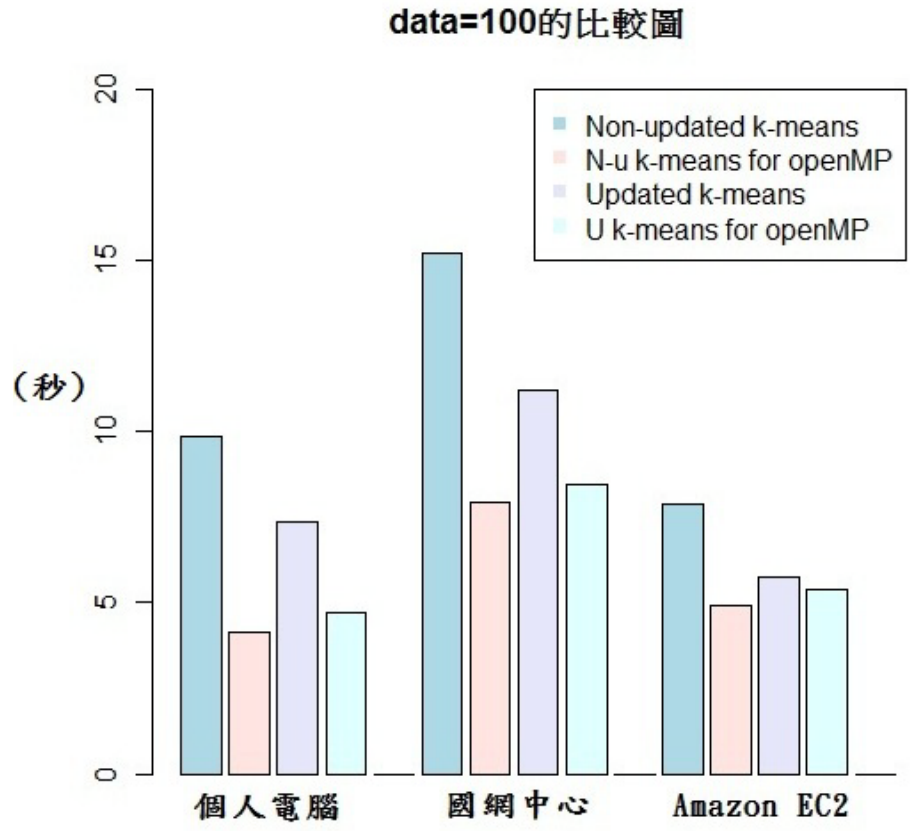
除此之外，圖十二的 Non-updated k-means 分群方法，因為遞迴次數太多，造成記憶體不足，而圖十五的 Non-updated k-means 分群方法，因為觀測值過多，也會造成記憶體不足，因此若要跑大資料量的程式，還是必須增加記憶體或者使用國網中心與 Amazon EC2!

	Linux 版本	CPU 規格	記憶體容量
個人電腦	Fedora 12	AMD Opteron 275 Dual Core (4 CPU)	2GB
國網中心 Formosa	Fedora	Intel(R) Core(TM) 2 Quad CPU (4 CPU)	8GB
雲端運算 Amazon EC2	Ubuntu 8.10	EC2 Compute Units (4 CPU)	15GB

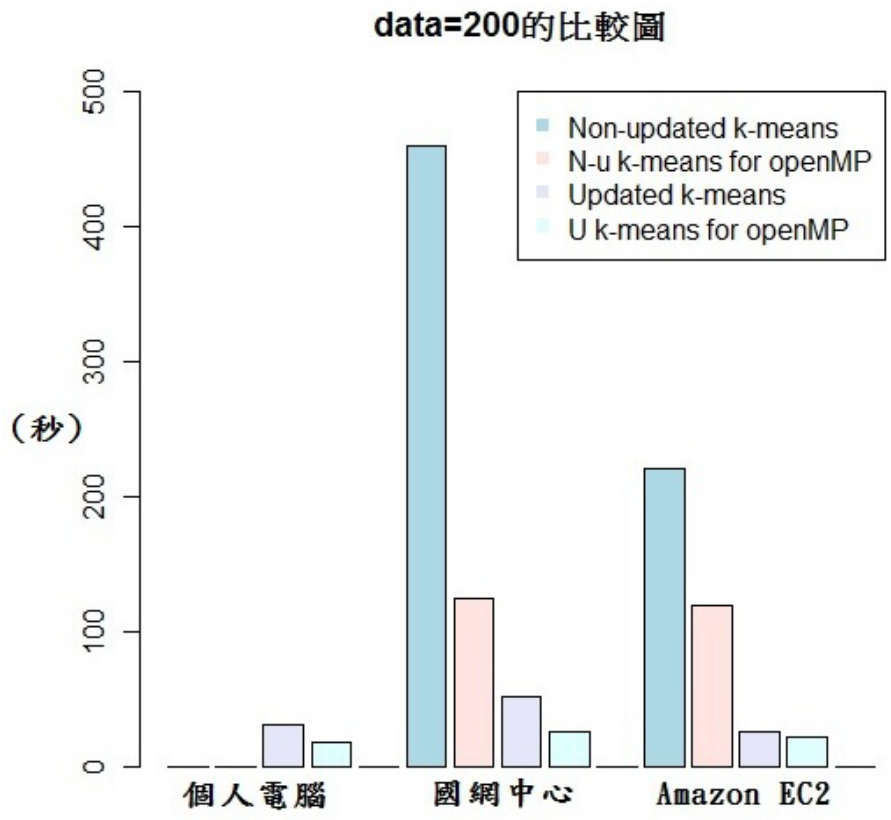
表九. 三種不同電腦環境的硬體規格



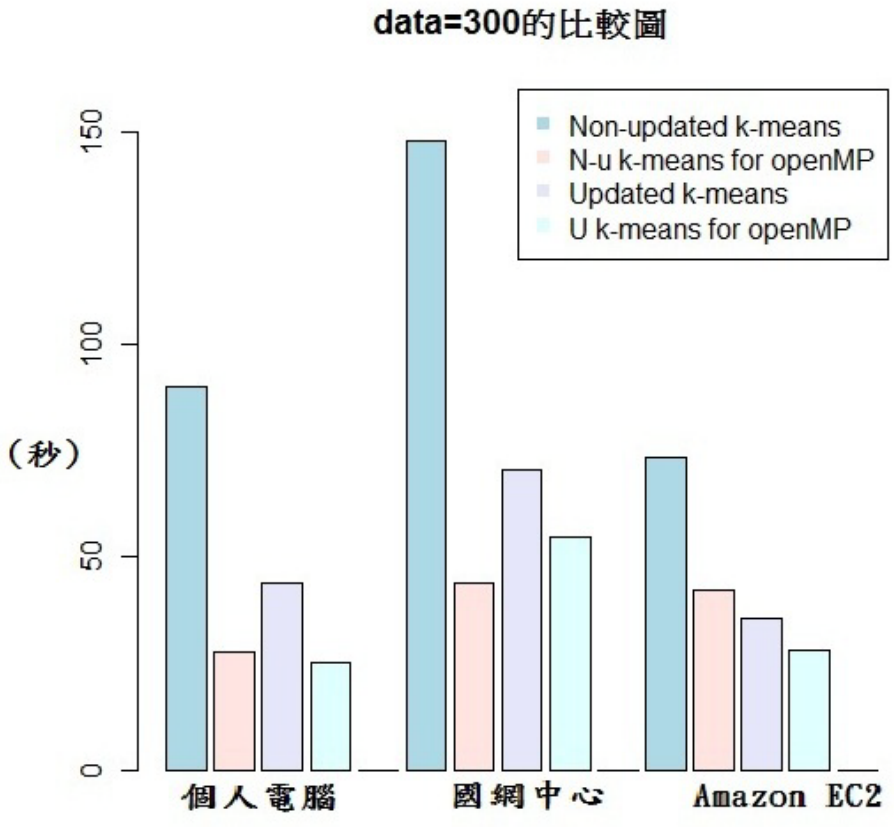
圖十.使用 openMP 且 data=70 的運行時間圖



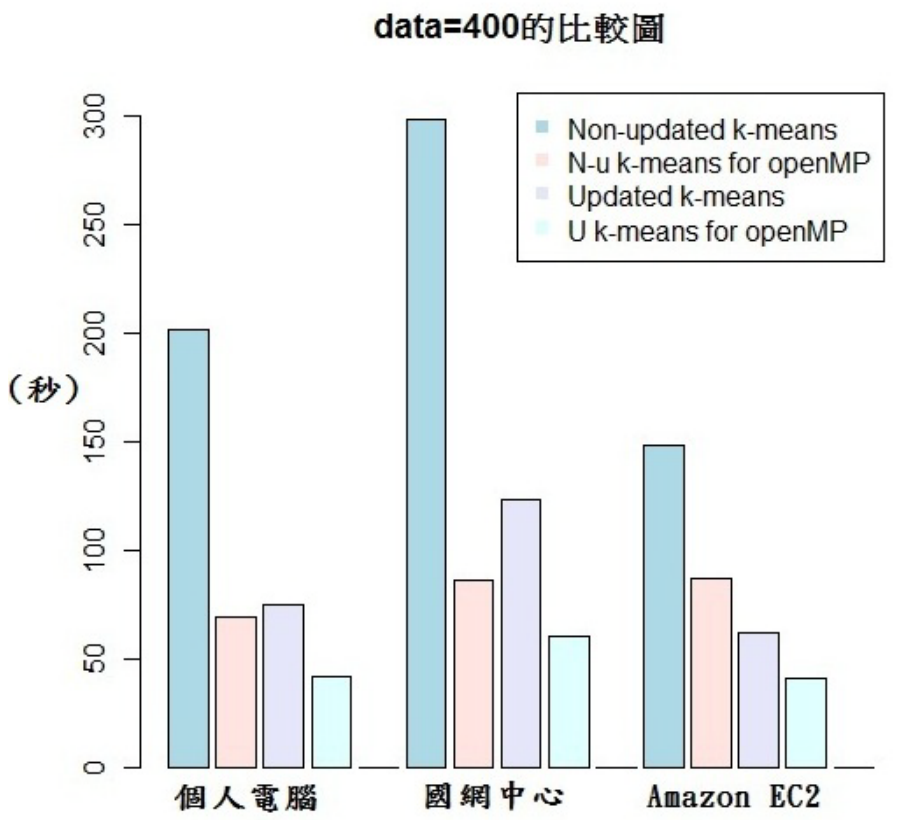
圖十一.使用 openMP 且 data=100 的運行時間圖



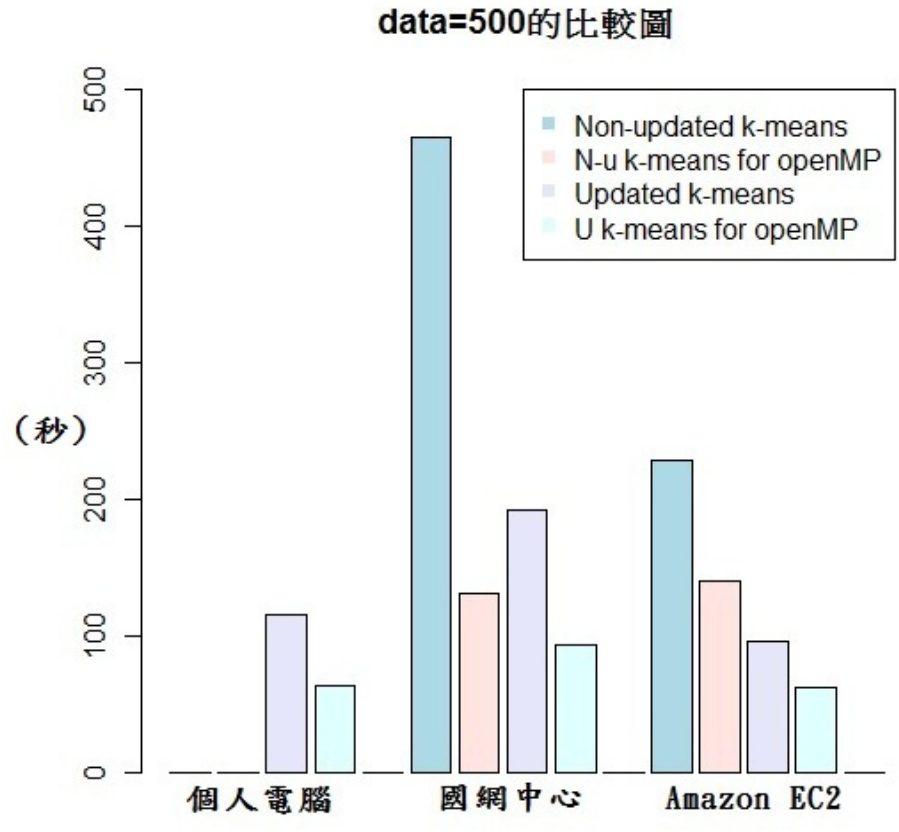
圖十二.使用 openMP 且 data=200 的運行時間圖



圖十三.使用 openMP 且 data=300 的運行時間圖



圖十四.使用 openMP 且 data=400 的運行時間圖



圖十五.使用 openMP 且 data=500 的運行時間圖

7. 討論

在這章節中，我們將討論在 Amazon EC2 虛擬電腦上架設 MPI 的問題以及在執行 k-means 分群方法估計潛在群體程式作 MPI 平行運算的問題。

在 4.2.5 章節中，介紹到在 Amazon EC2 虛擬電腦上架設 MPI 的程序，在第 2 步驟安裝與設定 NFS 的地方，我們不能順利執行 master@mpi-2:~$ sudo mount -t nfs mpi-1:/mpi /mpi 這步指令，其原因可能是在兩台主機防火牆的設定出現問題，但研究過後，mpi-2 無法順利通過 mpi-1 的防火牆，進而加載 mpi-1 的/mpi/mpi 目錄夾，導致後面程序無法執行。

在 4.1.3 章節中，介紹到使用 MPI 平行運算的運行流程，本論文使用的 k-means 估計潛在群體程式中，主要流程為 c 檔=>R 檔=>R 的 exe 檔，在 c 檔中，我們加入的 MPI 平行運算的程式，但因為 c 檔裡只有副函式，並沒有 main()函式，無法順利使用 MPI_Init()這個函數，導致 MPI 平行運算不能執行。

我們使用兩種方式去解決其問題，第一種方式的流程為 c 檔=>R 檔=>R 的 exe 檔=>MPI_C 的 exe 檔，在 MPI_C 的執行檔中，我們使用了 main()函式，也能順利呼叫 MPI_Init()，並且呼叫 R 的執行檔，希望在運行 R 的時候能在 c 檔中執行 MPI 平行運算，但還是在 c 檔使用 MPI 平行運算中無法使用 MPI 執行命令，其原因有可能是 C 呼叫 R 再呼叫 C 的過程中，因為中間有經過 R 的環境後，導致 MPI 的指令無法執行。

第二種方式流程為 c 檔=>R 檔=>R 的 exe 檔，在 R 檔中，我們先呼叫 library(Rmpi)，希望能開啟 MPI 平行運算並且在 c 檔中執行，此外，Rmpi 是一個能在 R 的環境下執行 MPI 的套件，但呼叫 Rmpi 後，還是會出現無法執行 MPI 的指令！

8. 結論

從上述的例子得知，使用 OpenMP 平行運算的程式，在程式執行時間上皆有明顯的減少，Non-updated k-means 分群方法雖然可以簡單地將樣本分組給其他電腦節點分群，但因為沒有立即的更新群體結構，造成遞迴次數較多，也影響了時間執行效率，而 Updated k-means 分群方法雖然不能直接將樣本分組給其他電腦節點，但也能在計算樣本變異數矩陣作平行運算，還是能縮短程式的執行時間，這對處理高維度的資料是有相當大的幫助，只要去除程式或演算法的相依性，就能簡單的加入平行運算語法，達到最佳的效率。

除此之外，我們還在三個不同的電腦環境運行程式，若沒有相關的電腦配備也能直接在國網中心或者 Amazon EC2 上執行程式，若有個人電腦的配備，只要有 gcc4.2 以上的版本，也能輕鬆的執行平行運算程式!

參考文獻

Amazon EC2 使用操作筆記(使用 Elasticfox)

http://plog.longwin.com.tw/my_note/2009/02/12/amazon-ec2-build-op-elasticfox-note-2009

Amazon web services

http://aws.amazon.com/

Bandeen-Roche, K., Miglioretti, D. L., Zeger, S. L., & Rathouz, P. J. (1997). Latent variable regression for multiple outcomes. Journal of the American Statistical Association, 92 , 1375-1386.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B, 39 , 1-38.

Goodman, L. A. (1974). Exploratory latent structure analysis using both identifiable and unidentifiable models. Biometrika, 61 , 215-231.

Huang, G. H., & Bandeen-Roche, K. (2004). Building an identifiable latent class model with covariate effects on underlying and measured variables. Psychometrika, 69 , 5-32.

Huang, G. H., Wang, S.M., & Hsu, C.C. Prediction of Underlying Latent Classes via K-means and Hierarchical Clustering Algorithms. Manuscript.

Kraj, P., Sharma, A., Garge, N., Podolsky, R., & McIndoe, R. A. (2008). ParaKMeans: Implementation of a parallelized K-means algorithm suitable for general laboratory use. BMC Bioinformatics, 9, 200.

Lazarsfeld, P. F., & Henry, N. W. (1968). Latent structure analysis. New York: Houghton-Mifflin.

Marc Snir & Steve W. Otto & Steven Huss-Lederman & David W.walker & Jack Dongarra (1996). MPI：The Complete Reference.

Message passing interface

http://www.mcs.anl.gov/research/projects/mpi/

Moustaki, I. (1996). A latent trait and a latent class model for mixed observed variables. British Journal of Mathematical and Statistical Psychology, 49, 313-334.

Official Python Planet@Taiwan

Setting Up an MPICH2 Cluster in Ubuntu 8.04 LTS Server(Quick-Note)
http://planet.python.org.tw/planet/tag/gnu/

OpenMP in Visual C++
http://msdn.microsoft.com/en-us/library/tt15eb9t.aspx

OpenMP.org
http://openmp.org/wp/

Peter S.Pacheco (1997). Parallel programming with MPI.

Rasch, G. (1960). Probabilistic models for some intelligence and attainment tests. Kopenhagen: Nielsen & Lydiche.

The R Project for Statistical Computing
http://www.r-project.org/

Titterington, D. M., Smith, A. F., & Makov, U. E. (1985). Statistical analysis of finite mixture distributions. New York: John Wiley & Sons.

William Group & Ewing Lusk & Anthony Skjellum (1999). Using MPI : Portable paralell programming with the message-passing interface.

國研院國網中心
https://www.nchc.org.tw/tw/

附錄

附錄一. 使用 openMP 且 data=70 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=70			
Non-updated k-mean	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **5.071607** secs user system elapsed 4.997 0.075 5.072 Time difference of **5.030866** secs user system elapsed 4.959 0.071 5.031	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **7.486753** secs user system elapsed 7.420 0.056 7.487 Time difference of **7.477474** secs user system elapsed 7.420 0.056 7.478	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **3.862151** secs user system elapsed 3.820 0.040 3.862 Time difference of **3.863628** secs user system elapsed 3.777 0.090 3.864
Non-updated k-mean for openMP	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **1.958733** secs user system elapsed 5.177 0.096 1.959 Time difference of **2.103989** secs user system elapsed 5.424 0.104 2.104	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **2.652442** secs user system elapsed 7.292 0.096 2.653 Time difference of **2.646764** secs user system elapsed 7.292 0.096 2.647	kmeans interations= 6 3 classes of sizes 39, 23, 16 Time difference of **2.350198** secs user system elapsed 3.709 0.118 2.350 Time difference of **2.547828** secs user system elapsed 3.808 0.069 2.547
Updated k-mean	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **3.437728** secs user system elapsed 3.367 0.069 3.438 Time difference of **3.441851** secs user system elapsed 3.375 0.066 3.442	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **5.536404** secs user system elapsed 5.500 0.032 5.537 Time difference of **5.529854** secs user system elapsed 5.476 0.048 5.530	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **2.891682** secs user system elapsed 2.824 0.070 2.891 Time difference of **2.88459** secs user system elapsed 2.861 0.030 2.885
Updated k-mean for openMP	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **2.154437** secs user system elapsed 6.099 0.050 2.155 Time difference of **2.280044** secs user system elapsed 6.303 0.075 2.280	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **3.098882** secs user system elapsed 5.832 0.092 3.099 Time difference of **3.141108** secs user system elapsed 5.956 0.112 3.141	kmeans interations= 4 3 classes of sizes 39, 23, 16 Time difference of **2.499721** secs user system elapsed 2.660 0.053 2.500 Time difference of **2.493003** secs user system elapsed 3.148 0.070 2.493

附錄二. 使用 openMP 且 data=100 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=100			
Non-updated k-mean	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **9.98927** secs user system elapsed 9.764 0.107 9.989 Time difference of **9.792687** secs user system elapsed 9.659 0.133 9.792	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **15.22142** secs user system elapsed 15.072 0.120 15.221 Time difference of **15.21041** secs user system elapsed 15.068 0.116 15.210	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **7.980843** secs user system elapsed 7.663 0.070 7.981 Time difference of **7.774865** secs user system elapsed 7.698 0.070 7.775
Non-updated k-mean for openMP	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **4.083199** secs user system elapsed 10.373 0.163 4.083 Time difference of **4.236732** secs user system elapsed 10.588 0.168 4.237	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **7.295465** secs user system elapsed 14.960 0.132 7.295 Time difference of **8.538599** secs user system elapsed 14.904 0.160 8.539	kmeans interations= 6 3 classes of sizes 38, 24, 16 Time difference of **4.787725** secs user system elapsed 7.667 0.135 4.787 Time difference of **5.022707** secs user system elapsed 7.560 0.138 5.023
Updated k-mean	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **6.88574** secs user system elapsed 6.803 0.081 6.886 Time difference of **7.866901** secs user system elapsed 7.764 0.100 7.867	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **11.19354** secs user system elapsed 11.128 0.052 11.193 Time difference of **11.20091** secs user system elapsed 11.084 0.100 11.201	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **5.762734** secs user system elapsed 5.691 0.080 5.763 Time difference of **5.759904** secs user system elapsed 5.671 0.090 5.760
Updated k-mean for openMP	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **4.653305** secs user system elapsed 12.966 0.097 4.654 Time difference of **4.743853** secs user system elapsed 13.179 0.093 4.744	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **8.331152** secs user system elapsed 11.648 0.108 8.331 Time difference of **8.563855** secs user system elapsed 11.640 0.128 8.564	kmeans interations= 4 3 classes of sizes 38, 24, 16 Time difference of **5.47492** secs user system elapsed 5.577 0.085 5.475 Time difference of **5.275207** secs user system elapsed 6.463 0.118 5.275

附錄三. 使用 openMP 且 data=200 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=200			
Non-updated k-mean		kmeans interations= 51 3 classes of sizes 39, 22, 17 Time difference of **7.683493** mins user system elapsed 454.700 5.572 461.009 Time difference of **7.662514** mins user system elapsed 454.032 5.244 459.751	kmeans interations= 51 3 classes of sizes 39, 22, 17 Time difference of **3.66743** mins user system elapsed 217.445 2.600 220.046 Time difference of **3.677179** mins user system elapsed 217.623 2.800 220.630
Non-updated k-mean for openMP		kmeans interations= 51 3 classes of sizes 39, 22, 17 Time difference of **2.082480** mins user system elapsed 447.632 8.620 124.949 Time difference of **2.076698** mins user system elapsed 447.551 8.468 124.601	kmeans interations= 51 3 classes of sizes 39, 22, 17 Time difference of **2.031848** mins user system elapsed 219.059 6.167 121.911 Time difference of **1.934487** mins user system elapsed 218.909 6.437 116.070
Updated k-mean	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **32.02866** secs user system elapsed 31.679 0.340 32.029 Time difference of **28.96982** secs user system elapsed 28.630 0.336 28.969	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **51.4201** secs user system elapsed 50.519 0.640 51.420 Time difference of **51.06895** secs user system elapsed 50.527 0.492 51.069	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **25.14705** secs user system elapsed 24.822 0.330 25.147 Time difference of **25.16046** secs user system elapsed 24.742 0.369 25.161
Updated k-mean for openMP	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **17.19106** secs user system elapsed 57.020 0.306 17.191 Time difference of **17.06588** secs user system elapsed 56.889 0.316 17.066	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **25.7224** secs user system elapsed 52.875 0.604 25.722 Time difference of **25.6091** secs user system elapsed 52.731 0.636 25.609	kmeans interations= 5 3 classes of sizes 40, 21, 17 Time difference of **20.30121** secs user system elapsed 24.512 0.207 20.302 Time difference of **23.76982** secs user system elapsed 24.595 0.110 23.770

附錄四.使用 openMP 且 data=300 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=300			
Non-updated k-mean	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **1.396728** mins user system elapsed 82.543 1.233 83.804 Time difference of **1.61341** mins user system elapsed 95.426 1.351 96.804	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **2.470043** mins user system elapsed 146.841 1.336 148.202 Time difference of **2.469977** mins user system elapsed 146.805 1.368 148.199	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **1.227424** mins user system elapsed 72.079 1.550 73.645 Time difference of **1.228795** mins user system elapsed 72.101 1.630 73.727
Non-updated k-mean for openMP	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **27.69056** secs user system elapsed 88.389 2.088 27.690 Time difference of **27.61994** secs user system elapsed 88.251 2.042 27.620	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **43.6328** secs user system elapsed 145.121 2.860 43.633 Time difference of **44.03856** secs user system elapsed 145.225 3.616 44.038	kmeans interations= 7 3 classes of sizes 29, 32, 17 Time difference of **44.03862** secs user system elapsed 72.050 2.361 44.038 Time difference of **40.19318** secs user system elapsed 73.441 1.984 40.194
Updated k-mean	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **40.7871** secs user system elapsed 40.153 0.623 40.788 Time difference of **47.2784** secs user system elapsed 46.525 0.731 47.278	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **1.173537** mins user system elapsed 69.696 0.700 70.413 Time difference of **1.172719** mins user system elapsed 69.828 0.520 70.363	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **35.5649** secs user system elapsed 34.739 0.822 35.565 Time difference of **35.54702** secs user system elapsed 34.853 0.700 35.547
Updated k-mean for openMP	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **24.266** secs user system elapsed 78.571 0.525 24.266 Time difference of **26.04723** secs user system elapsed 85.350 0.493 26.047	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **53.53604** secs user system elapsed 73.168 0.992 53.536 Time difference of **56.20978** secs user system elapsed 73.156 0.924 56.209	kmeans interations= 3 3 classes of sizes 29, 32, 17 Time difference of **30.72546** secs user system elapsed 36.445 0.506 30.726 Time difference of **25.82094** secs user system elapsed 39.227 0.628 25.820

附錄五. 使用 openMP 且 data=400 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=400			
Non-updated k-mean	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **3.300604** mins user system elapsed 180.515 3.473 198.036 Time difference of **3.403863** mins user system elapsed 182.601 3.348 204.231	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **4.970054** mins user system elapsed 295.206 2.988 298.204 Time difference of **4.970812** mins user system elapsed 295.158 3.088 298.249	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **2.466033** mins user system elapsed 144.754 3.200 147.962 Time difference of **2.464888** mins user system elapsed 144.673 3.221 147.894
Non-updated k-mean for openMP	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **1.148515** mins user system elapsed 174.785 6.447 68.911 Time difference of **1.162474** mins user system elapsed 174.033 5.958 69.749	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **1.435571** mins user system elapsed 293.562 7.104 86.134 Time difference of **1.439072** mins user system elapsed 293.930 6.884 86.344	kmeans interations= 8 3 classes of sizes 26, 35, 17 Time difference of **1.420487** mins user system elapsed 145.462 4.728 85.229 Time difference of **1.481086** mins user system elapsed 145.597 4.829 88.865
Updated k-mean	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **1.313125** mins user system elapsed 77.373 1.348 78.788 Time difference of **1.179180** mins user system elapsed 69.459 1.275 70.751	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **2.048304** mins user system elapsed 121.607 1.276 122.899 Time difference of **2.049078** mins user system elapsed 121.643 1.280 122.945	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **1.032571** mins user system elapsed 60.879 1.070 61.954 Time difference of **1.031569** mins user system elapsed 60.531 1.361 61.894
Updated k-mean for openMP	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **41.5224** secs user system elapsed 139.532 2.434 41.523 Time difference of **42.23299** secs user system elapsed 139.404 3.128 42.233	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **1.006742** mins user system elapsed 126.651 1.244 60.404 Time difference of **1.009029** mins user system elapsed 126.435 1.316 60.542	kmeans interations= 3 3 classes of sizes 26, 35, 17 Time difference of **39.32278** secs user system elapsed 63.346 0.353 39.323 Time difference of **42.40546** secs user system elapsed 68.551 1.365 42.405

附錄六.使用 openMP 且 data=500 的運行時間表

	個人電腦	國網中心 Formosa II	Amazon EC2
Data=500			
Non-updated k-mean		kmeans interations= 8 3 classes of sizes 27, 37, 14 Time difference of **7.761284** mins user system elapsed 460.300 5.336 465.677 Time difference of **7.75623** mins user system elapsed 460.084 5.292 465.374	kmeans interations= 8 3 classes of sizes 27, 37, 14 Time difference of **3.816521** mins user system elapsed 224.086 4.910 228.991 Time difference of **3.81692** mins user system elapsed 224.284 4.730 229.015
Non-updated k-mean for openMP		kmeans interations= 8 3 classes of sizes 27, 37, 14 Time difference of **2.170563** mins user system elapsed 449.708 8.400 130.233 Time difference of **2.179763** mins user system elapsed 449.616 8.764 130.786	kmeans interations= 8 3 classes of sizes 27, 37, 14 Time difference of **1.911909** mins user system elapsed 226.075 7.852 114.715 Time difference of **2.732940** mins user system elapsed 224.506 9.134 163.977
Updated k-mean	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **1.850365** mins user system elapsed 109.061 1.934 111.022 Time difference of **2.003627** mins user system elapsed 118.152 2.045 120.217	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **3.213469** mins user system elapsed 190.531 2.268 192.808 Time difference of **3.20747** mins user system elapsed 190.259 2.164 192.448	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **1.593715** mins user system elapsed 93.766 1.860 95.623 Time difference of **1.595637** mins user system elapsed 93.716 2.020 95.738
Updated k-mean for openMP	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **1.046877** mins user system elapsed 214.635 3.545 62.813 Time difference of **1.058719** mins user system elapsed 215.512 4.238 63.523	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **1.545340** mins user system elapsed 196.208 1.784 92.721 Time difference of **1.549201** mins user system elapsed 196.448 1.716 92.952	kmeans interations= 3 3 classes of sizes 27, 34, 17 Time difference of **1.056813** mins user system elapsed 106.173 1.642 63.409 Time difference of **1.010068** mins user system elapsed 106.014 1.744 60.604