國立交通大學

生醫工程研究所

碩 士 論 文

FGK 動態壓縮演算法

支援分散處理系統實做

FGK dynamic compression algorithm support distributed

processing implementation of the system

研 究 生：洪崇誠

指導教授：荊宇泰　教授

中 華 民 國　九 十 九 年 十一 月

FGK 動態壓縮演算法支援分散處理系統實做

FGK dynamic compression algorithm support distributed processing implementation of the system

研 究 生：洪崇誠　　Student：Chung-Cheng Hung

指導教授：荊宇泰　　Advisor：Yu-Tai Ching

國 立 交 通 大 學

生 醫 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Biomedical Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年十一月

FGK 動態壓縮演算法支援分散處理系統實做

研 究 生：洪崇誠　　指導教授：荊宇泰

國 立 交 通 大 學

生 醫 工 程 研 究 所

摘要

我們的研究目的是把一個複雜的運算，把它切割成較小的運算來做處理，而較小的運算間再各自交給自己的處理器來做處理，如此可以免去單一運算所需耗費的冗長的時間，而我們則是採取動態壓縮的演算法來面對我們傳輸所遇到的問題，此種演算法有隨壓隨解的性質，可以先把解壓縮過後的資料來拿先做處理，可以省去較多時間。

而我們還採取了網路傳輸的一些觀念，模擬多核心的架構，我們運用了 3~5 台的電腦來實做我們的系統架構，來達到我們的實驗需求，已獲得合理且有效的實驗數據。

FGK dynamic compression algorithm support distributed processing implementation of the system

Student：Chung-Cheng Hung Advisor：Yu-Tai Ching

Institute of Biomedical Engineering
National Chiao Tung University

Abstract

Our aim is to a complex operation, it cut less computation to do processing, and smaller operations in their respective rooms to deal with their own processor to do so without having a single operator can spend the necessary the long time, and we are taking the dynamic compression algorithms to transmit the face of our problems, this algorithm has properties of solutions with the pressure as can be first to come and collect the data after decompression do first processing, can save more time.

And we have also taken some ideas network transmission, analog multi-core architecture, we use the computer at 3 to 5 we implement our system architecture, to achieve our experimental needs, has been given a reasonable and effective experimental data .

致謝

在兩年多的日子裡，有著同學間彼此的照顧，實驗室間彼此的學習交流與成長，讓我能在這段期間獲得良好的知識，與在交大這良好的學習環境中成長，感謝教授的細心指導，批改與督促學生相關論文的完成與進度，感謝實驗室同學間彼此的互相鼓勵，感謝清大的同學在實驗上給予指導和幫助，感謝台大的學長在論文上的指點與指教，感謝中央的同學在修課上的幫助，在這些日子裡都有你們的回憶在我心中，最後感謝家人在精神上與經濟上的支持，讓我能在這段日子中有強烈的支持，讓我能在良好的學習環境中成長，在這裡感謝在這段期間有給我溫暖的每一個師長、同學，謝謝你們 !

目錄

中文摘要 ……ii

英文摘要 ……iii

致謝 ……iv

目錄 ……v

第一章　緒論 ……1

1.1 簡介……1

1.2 論文架構……1

第二章　背景與理論基礎……2

2.1　前言……2

2.2　平行計算……3

2.3　Huffman 演算法介紹……5

2.4　FGK 演算法介紹……6

2.5　Gaussian filter 介紹……13

第三章　實驗實作流程……14

3.1　實驗簡介……14

3.2　FGK 演算法……14

3.3　程式平行處理部分……20

3.4　檔案一切四和四合一部分……22

3.5　網路傳輸部分……23

3.6　即時動態壓縮影像系統總覽……24

第四章　實驗結果分析……25
4.1　實驗簡介……25
4.2　實驗環境……25
4.3　實驗結果……26
第五章　結論與未來展望……38
參考文獻……39

圖表目錄

圖 1.單一運算的架構系統圖……4
圖 2.平行計算的架構系統圖……4
圖 3.(a)(b)(c)　三個樹的轉換圖形……10
圖 4.輸入新的 symbol 之後 Dynamic Huffman tree 的轉換過程……11
圖 5.Gaussian filter 示意圖……13
圖 6.切割完後左上方的圖片……22
圖 7.國家同步輻射中心所提供之原始資料……26
圖 8.最後經過高斯處理回傳回來的圖檔……27
圖 9.第二筆原始的圖檔……28
圖 10.經過實驗過後產生的圖檔……29
圖 11.第三筆原始的圖檔……30
圖 12.經過實驗過後產生的圖檔……30
圖 13.上方表格三種執行時間之圖表整理……33
圖 14.上方表格三種運算的 speedup 之圖表整理……34
圖 15.上方表格四個 Recv 端 Gaussian 和 Total 執行時間之圖表整理……36
圖 16.上方表格整批檔案傳輸運算的 speedup 之圖表整理……36

第一章　緒論

1.1 簡介

平行運算為當今目前一個重要的用來加快速度的方法，而在本篇的論文中，我們探討的是當一張圖檔讀進來後，我們分別用 2 台到 4 台其它電腦當 server 端，來處理我們 client 端的這邊電腦傳檔案所需的部分。而平行處理的處理部分，我們也是採用把一個問題切割成若干份的方式來處理，之後再把切割後的圖檔部份分給各個處理器去執行。而在於提高效能的部分，我們則是採取了動態壓縮的演算法來做增進效能的部分，使用動態壓縮演算法的好處為可以在當一個檔案切割成若干份時，當若干檔案需要做通訊傳給各個處理器去處理時，可以有隨壓隨解的功能存在，即可在 client 端還在做處理傳輸時，server 端即可把先接收的資料拿來做事先的運算，如此在時間上即可節省許多多餘的等待時間的耗費，我們借由平行運算和動態演算法的幫助，讓我們的整體效能上能有實質的提升。

1.2 論文架構

本篇論文分為五個架構，第一章為緒論的部分，這部分主要是來介紹相關研究的發展與動機，和研究的目標和方向，第二章的部分，則會介紹我們的研究的相關的背景和資料的訊息，和一些相關的文獻回顧，把理論和相關的基礎簡介將會在此章節介紹，第三章的內容，則會把本篇論文的實驗方式，和實

驗流程，將在本章節做個詳細的介紹和統整，讓大家可以了解整個論文的實作細節與方法，第四章為實驗的一些結果的圖片和數據的分析，第五章則為結論及未來的展望。

第二章 背景理論與基礎

2.1 前言

處理檔案大的資料時，需要借用分散式系統來做處理，我們設計了一個這樣的系統，其中為了降低 processors 之間溝通的成本，所以用動態壓縮 的方法來降低時間，來加快速率。

FGK 演算法，原本一般常見的 Huffman 演算法，是採用 2-pass 的壓縮演算法的流程，必須把全部的檔案全部收集完後，就是每個檔案的字元都收集完成之後，此為第一階段，才能進行演算法的壓縮過程，此為第二階段，如此的演算法需耗費較龐大的時間來處理我們所需要的檔案，在這裡我們引用了 knuth 提出的 FGK 演算法來實做我們的實驗過程，FGK 演算法可以提供隨壓隨解的演算法流程，也就是壓縮完一個字元之後，隨即可把該字原來做解壓縮來處理，如此可以不需等到全部檔案壓縮完，即可把到目前壓縮的字元隨即壓縮出去，節省許多的時間。

所以本篇論文在提出了更新的方法，以原本的 FGK 架構為基礎，來處理面對較大檔案的處理問題，我們在加上多個 process 可以平行處理的架構，讓檔

案先切成較小的等份後，可以以平行處理的傳輸出去，這樣不僅可以達到隨壓隨解的效果，也可以加速傳輸的速度，來達到增加我們傳輸速度的目的與效果。

在本篇的論文中，提及了許多的演算法，以下將會把本篇論文所提及的一些基礎的演算法一 一的介紹，讓大家先有個基礎的概念。

2.2 平行計算

平行運算是資訊科學中的專有名詞。凡令多台伺服器、多台電腦、一台電腦中的多顆 CPU 一直到 CPU 中的多核心，同時進行同一項工作就稱為平行運算。過往這項技術應用在超級電腦上，通常用於軍事、氣候、物理等需要精密、快速的運算用途上。

以下先來看看如果只使用單一個 CPU 的架構圖，因為只使用單一的程序來執行，所以程序必需一步步的流程走完才能換下一步驟，如此可能會耗費許多大量的時間來做處理，因而有人提出了另一個平行處理的方法。

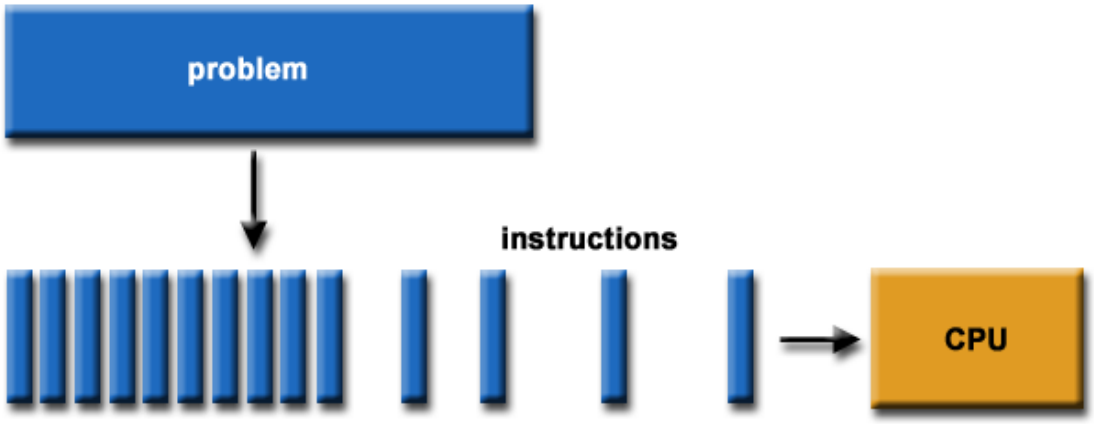


圖 1. 單一運算的架構系統圖

平行處理的方法如下圖所示，先切割成若干的等份後，再把這些等份分散到其他幾台處理器來做同樣動作的運算，如此類似平行化功能的方式，不會讓只有單一台電腦自己本身再做單一的運算，而是讓多 CPU 來做處理，此在解決大型問題上，是相當有效的方法，而且另外可即時性的獲取更多網路上的資源，而達到速度方面的提升的優點，但是記憶體容量的不足的部分，會是此架構產生的一個缺點。

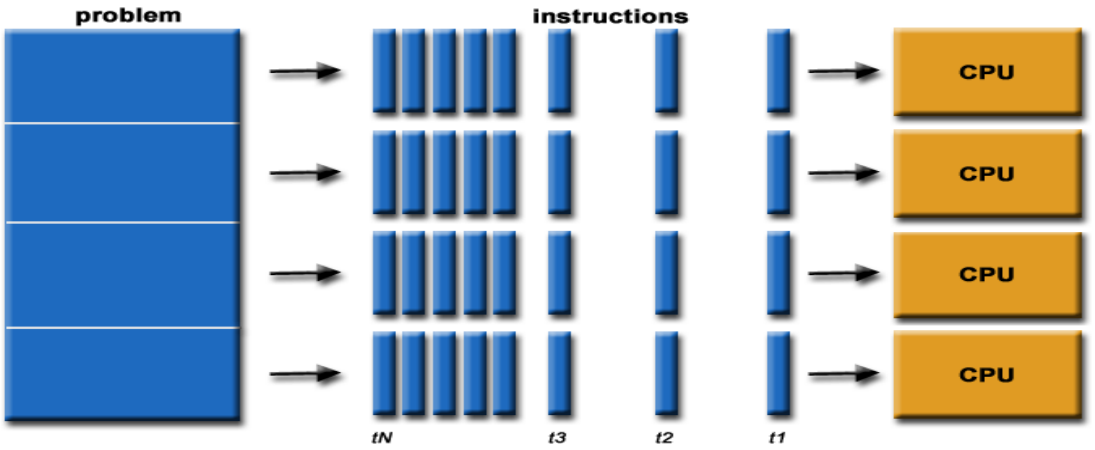


圖 2. 平行計算的架構系統圖

2.3 Huffman 演算法介紹

Huffman 演算法是 MIT 的 David Huffman 發明的。這個方法是當初他為了不想考研究所的期末考而著手挑戰的難題，而他想出的這個方法也成為最廣為人知的壓縮法之一。

Huffman 演算法法是由最底層的葉部開始建構起。Huffman 演算法如下：

1. 統計每個符號的出現機率，建立數個節點，每個節點包含一個符號和它的出現機率。
2. 將節點依機率大小排序好。
3. 將機率最小的兩個節點放在一起，並且產生一個父節點以做為一顆樹，父節點的權重相當於兩個子節點的數字合。
4. 將父節點視為新的節點排入原本的節點中考慮，原本的兩個子節點則不再考慮。
5. 原本的兩個子節點中一個指定二進位 1 的數字，一個指定 0 的數字。
6. 重複 2 到 5 的步驟，直到只剩下一個節點可以考慮，這個節點就是整顆編碼樹的根節點。

藉由以上的程序我們可以建構一顆編碼樹，其中每個樹葉節點都是我們資料中出現的符號，而從樹根走訪至樹葉所會經過的節點中指定的數字就是那個樹葉上的符號所用的編碼。我們簡單的計算一下這個演算法能壓縮資料的程度，假設有一筆資料經統計後含有 15 個 A、7 個 B、6 個 C、6 個 D 和 5 個 E 經由 Huffman

演算法編碼後，五個字母的字碼分別變為 0、100、101、110、111，而總共所花費的空間是 87bits。

2.4 FGK 演算法介紹

如果是原本傳輸的通訊則傳統的霍夫曼編碼必造成延遲現象，而且接收碼收端也需要額外的儲存空間，編解碼時需要對這個接收檔案作搜尋的動作，有時候也會影響整個演算法的效率，因此就有人提出一道手續就能完成的霍夫曼編碼，最早提出的是Faller 以及Gallager 這兩個人，稍後由Knuth 加以改良，統稱為FGK演算法，這個僅需一道手續的霍夫曼演算法在編碼時需送出 t+1 種符號(其中t為欲壓縮資料內部所擁有的符號種類數目)，額外的符號是為了要代表在編碼時即將要送出新的符號，所送的控制碼。在開始編碼以及傳送碼字的同時，傳送者以及接收者需先初始化霍夫曼樹，並且同步的去更新霍夫曼樹，當傳送端輸入一個符號並且準備傳送碼字出去時，需把剛剛輸入的符號丟入演算法內並且去修改霍夫曼樹，同時當接收端收到碼字並且解出符號時，也需和傳送端做相同修改霍夫曼樹的步驟，而所需要的符號則是剛剛解碼出來的符號，這樣一來傳送端不需送一筆霍夫曼解碼所需的碼簿給接收端，也因此這個新的演算法能夠節省碼簿所佔的額外空間，以下我們將介紹 Knuth 的演算法:

首先我們先定義一些係數:

n =符號所擁有的數量;

a_j = 在符號系統中第 j 個符號;

t =已處理過的字母數量;

u_t = a_{i1} a_{i2}…… a_{it}代表消息源中前t 個字母;

k =在霍夫曼編碼中已經使用到的符號的數量;

w_j =代表已編碼過的符號如a_j ，在編碼過程中被處理的次數;

l_j =在霍夫曼樹中a_j節點到根節點的距離;

其中 $1<=j、k>=n$ ， $0<=w_j<=t$;

動態霍夫曼樹還伴隨著以下的特徵:

成員特性:

(1) 霍夫曼樹的子節點擁有自己的權位值， $W_1, W_2, \ldots . W_b$ ，而且中間節點的權位值為兩個子節點的權位和。

(2) 當節點從下往上堆積成霍夫曼樹時，每個節點需被編號所以第2j-1 個節點與第2 j 個節點有著成員關係(代表兩者的父節點為同一個)，而其父節點的編號必大於兩個子節點。

這些已編號過的節點，我們必須利用編號的順序來建構整個霍夫曼樹，例如1 號節點與2 號節點先被合併在一起，接著3 號與4 號節點則在下次被合併在一起，接著是5 號和6 號節點… … 如此依序合併在一起，即可以建構成一個霍夫曼樹。

當建構一個動態霍夫曼樹時，如果有一個符號輸入並且準備要改變霍夫曼樹

底層子節點的權位時，而這個子節點的權位值也大於比他還高的節點之權位值時(越高代表越接近根節點)，如何快速的改變霍夫曼樹內部節點的位置，是個值得我們討論的地方。

舉個例子來說明我們如何快速改變霍夫曼樹節點的位置，如圖2.2-5 這張圖剛好是消息源已經被編了32 個符號(也就是t=32，t 為剛剛定義過的係數)，當第33個符號’b’ 從消息源要進來編碼時，如果只是把霍夫曼樹內的編號為2 的子節點以及祖先節點群的權位全部加1 的話，會發現編號為4 的節點其權位值為6，而同一層的的節點群3 號5 號6 號節點群其權位值分別為5,5,6，但是霍夫曼樹編碼的原理是把權位值最低的兩個節點合併在一起，依照這個原理的話應該是3號和5 號節點合併在一起，所以這樣做的結果並不符合霍夫曼樹的原理，解決的方法就是當新進節點進來後，立即尋找編號比他大且權位值相同的節點，一但找到就立即作旋轉的動作(即交換兩個節點所在的位置)，找不到則下一個欲交換位置節點為此節點10父節點(如果有交換則為新的父節點)，再做一次搜尋的動作，一但發現有比欲交換位置節點的編號還大且權位值相等的節點時就準備旋轉，但是此時則是旋轉以這個欲交換位置節點為根節點的小樹，之後找到可以旋轉的節點如果是中間節點也是旋轉以這個中間節點為根節點的小樹，重複尋找與交換的步驟直到欲交換位置節點為根節點，如圖 1.(a) 到圖 1.(b) 為交換後的結果，最後再把剛剛新進節點的權位值加1 並且依序往上移動權位值也依序加1，直到根節點為止如圖1.(b) 到圖 1.(c) 。

做完霍夫曼樹更新的動作後，我們可以發現節點依舊符合剛剛定義的成員特性，從圖 1.(c) 可以發現1 號節點和2 號節點被合併在一起，所以符合成員特性的條件1，也符合條件2 即父節點的編號大於其子節點的編號。

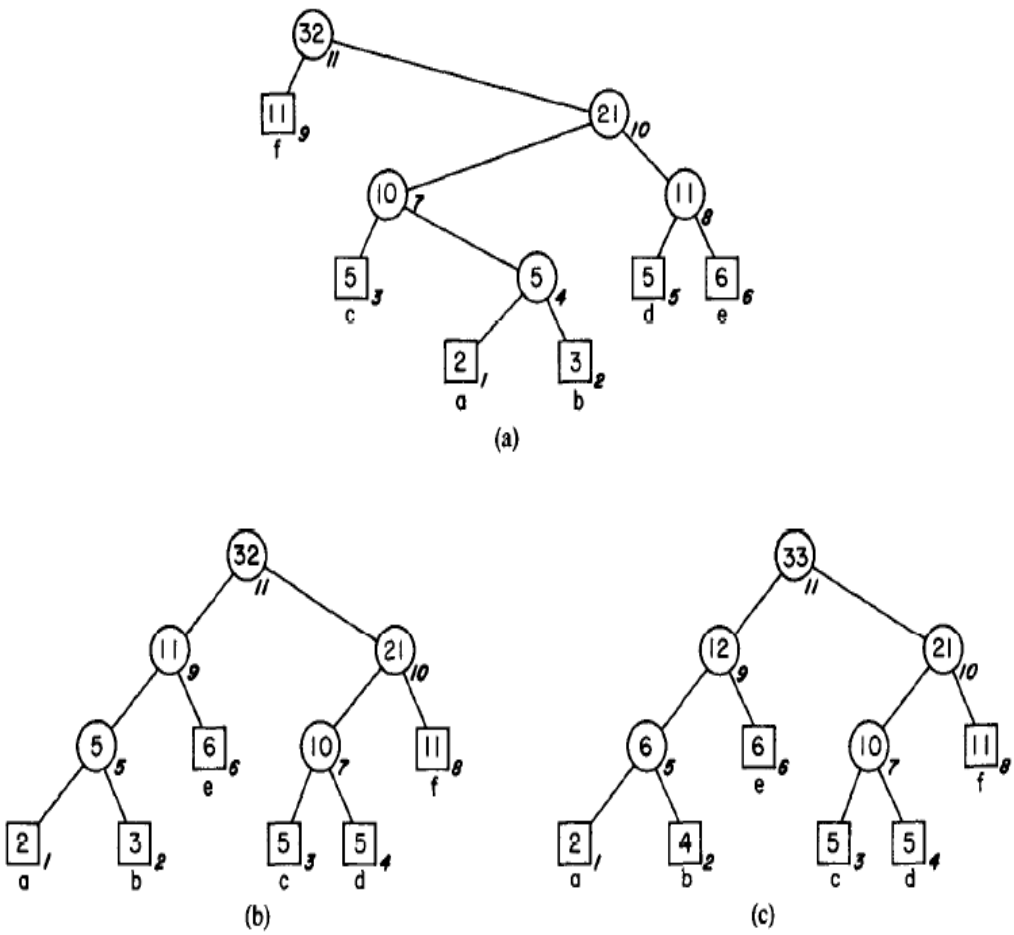


圖 3.(a)(b)(c)　三個樹的轉換圖形

在編碼的過程發現 k<n 時，代表還有未在霍夫曼樹出現過的符號，所以我們必須要多用一個權位值為0 的葉子節點來代表未出現的符號，在編碼過程時如果輸入一個新的符號時，就必須分裂剛剛權位值為0 的節點，並往下延伸出兩個葉子節點，其中左葉子節點依舊代表權位值為0 的節點，右葉子節點則代表為剛剛輸入的新符號如圖2，這時我們必須傳送未分裂前權位值為0 的節點所代表的

碼字，告訴接收端有一個新符號出現了，並送新符號在原本定義好的符號系統內所代表之索引值出去，這樣接收端也才知道是哪一個新符號要出現，圖3 為動態霍夫曼樹更新的一段虛擬碼。

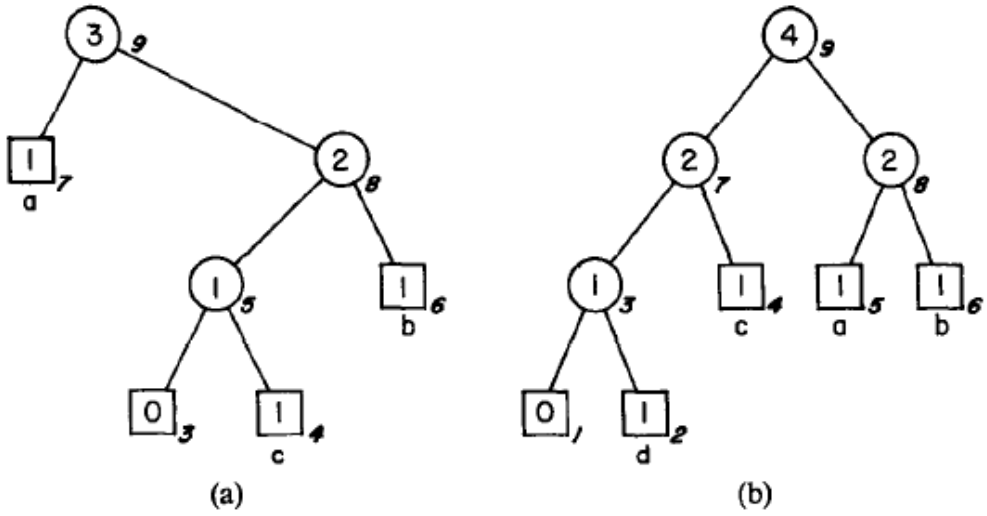


圖4. 輸入新的symbol之後Dynamic Huffman tree的轉換過程

在圖3 的虛擬碼中可以看出，當新的節點q 進來且必須和其他節點作交換時，找到比q 節點還高的節點時則做往上交換的動作，同一高度做往右交換的動作，如圖1 的a 圖到b 圖可以得知節點8 和節點9 是做往上交換的動作，節點4 和節點5 是做往右換的動作。

```
procedure Update;
begin
q := leaf node corresponding to a_{i+1};
if (q is the 0-node) and (k < n − 1) then
   begin
   Replace q by a parent 0-node with two leaf 0-node children, numbered in the order left
      child, right child, parent;
   q := right child just created
   end;
if q is the sibling of a 0-node then
   begin
   Interchange q with the highest numbered leaf of the same weight;
   Increment q's weight by 1;
   q := parent of q
   end;
while q is not the root of the Huffman tree do
   begin   {Main loop}
   Interchange q with the highest numbered node of the same weight;
   {q is now the highest numbered node of its weight}
   Increment q's weight by 1;
   q := parent of q
   end
end;
```

2.5 Gaussian filter 介紹

$$G(x,y)=\frac{1}{2\pi\sigma}\exp\left[\frac{-(x^2+y^2)}{2\sigma^2}\right] \quad (1)$$

(1)式為 Gaussian 的 數學函數式子，而我們運用了這個式子的轉換之後，得到了一些 Gaussian 的 Mask ，而這些 Mask 就是所稱的 Gaussian filter ，而這次實驗我們就是以 Gaussian filter 來做對圖片的處理，以下為 Gaussian filter 的轉換示意圖:

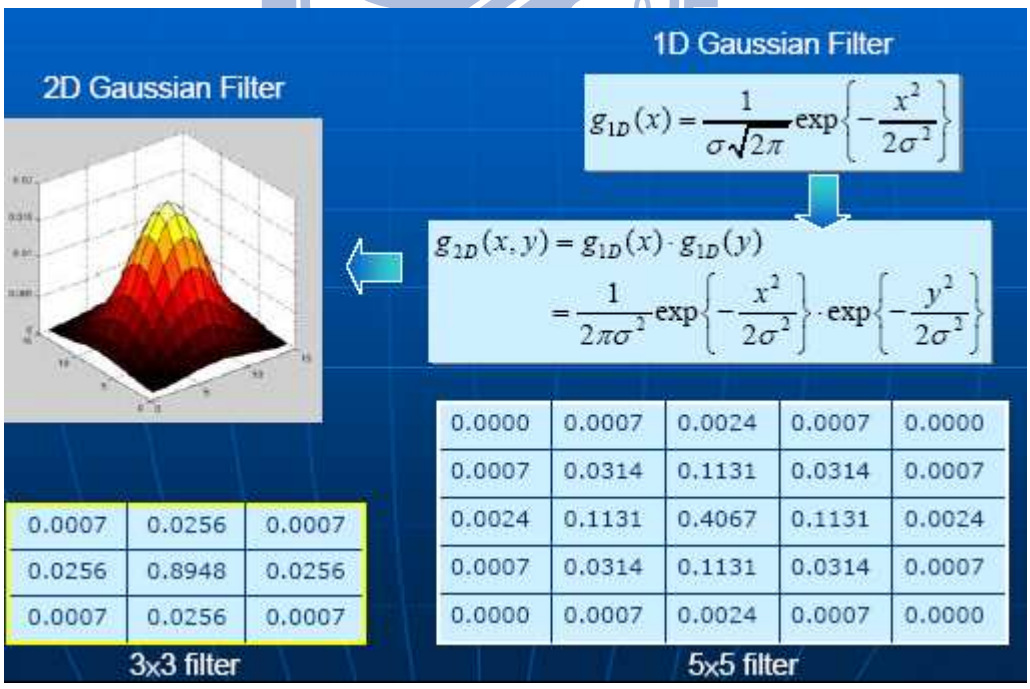


圖 5. Gaussian filter 示意圖 來源： [徐百輝,2004]

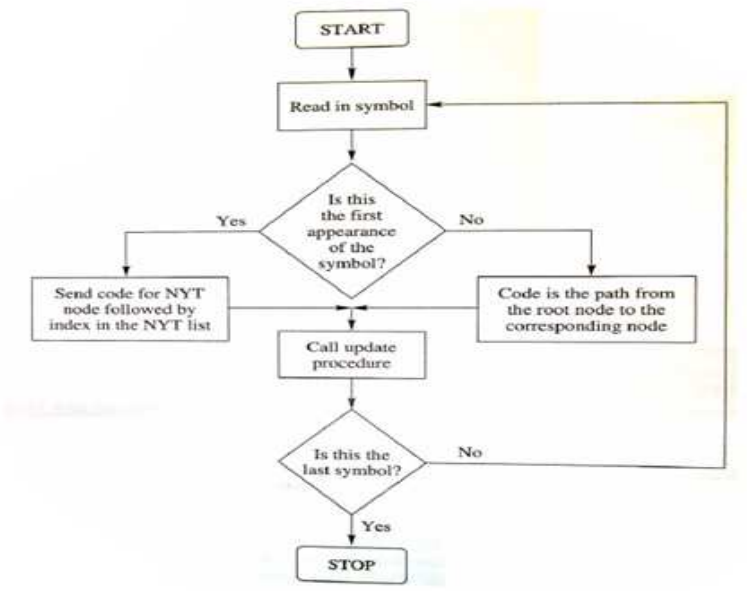
第三章　實驗實作流程

3.1 實驗簡介

在本章中，我們利用了第二章所提到的一些背景基礎為架構，來實做壓縮演算法的平行處理方法，使得資料檔案，可以在較短的時間內，經由 FGK 演算法處理後，隨即把檔案解出，不僅可以節省時間，也可讓資料在較短的時間內讓對方就可收到訊息，不會流失，達到隨壓隨解的功能，我們用到了一些網路傳輸和多個 process 同時傳輸的觀念，讓檔案先經過切割後，先減少傳輸時間，再把四個檔案所呼叫的程式同時 create 起來，讓四個檔案可以同時來做平行處理，之後再把檔案經過高斯處理，讓資料可以在送到收端時，可以在收端的時候，做一些運算之後，再回傳給送端處理完之後的檔案，之後再把四個切割處理後的檔案，同樣也是以隨壓隨解的方式從收端傳回給送端的檔案，最後把四個小的切割後的圖檔還原合成一個大的原始經過高斯處理後的圖檔，此即為我們的實驗流程。

3.2 FGK 演算法

FGK 演算法是 Kunth 所提出的演算法，主要是兩個部分，一個是 Encode 的部分，即為檔案傳進來的時候壓縮的部分，以下為壓縮部分的整個流程:



Encode 可以分為兩部分來討論:

1. 如果檔案壓縮的字元,為第一次出現的檔案,就把該字元產生對應的 NYT node 編碼的 short code 編出，而 NYT 而傳輸的過程為(1)先傳從 leaf 到 root 的路徑,即為到 NYT node 的路徑,(2)傳完之後再把該字元所產生的 short code 傳出，(3)傳完再經過 update 把樹做更新的動作，此為檔案中該字元第一次出現時所傳輸的檔案順序流程。

2. 如果檔案遇到壓縮的字元已經出現過的時候,例:字元 A 已經在之前檔案中已經傳輸過的時候，它會跑以下的程序流程,(1)直接把該點所在當時後的樹的的 code 傳出(2)之後傳完之後再跑 update 把樹做更新的動作,此即為遇到 z 已經壓縮過後的字元，程式處理流程的順序。

3. Update 部分為樹做更新的動作，以下為它的程式部分，主要就是把樹的字元權重出現較多的，把它移到樹較高的位置，讓整個傳輸的 code 數可以較少，此為此 FGK 演算法主要想法，而每壓縮一個 letter 數，樹的形狀就會跟著改變一次:

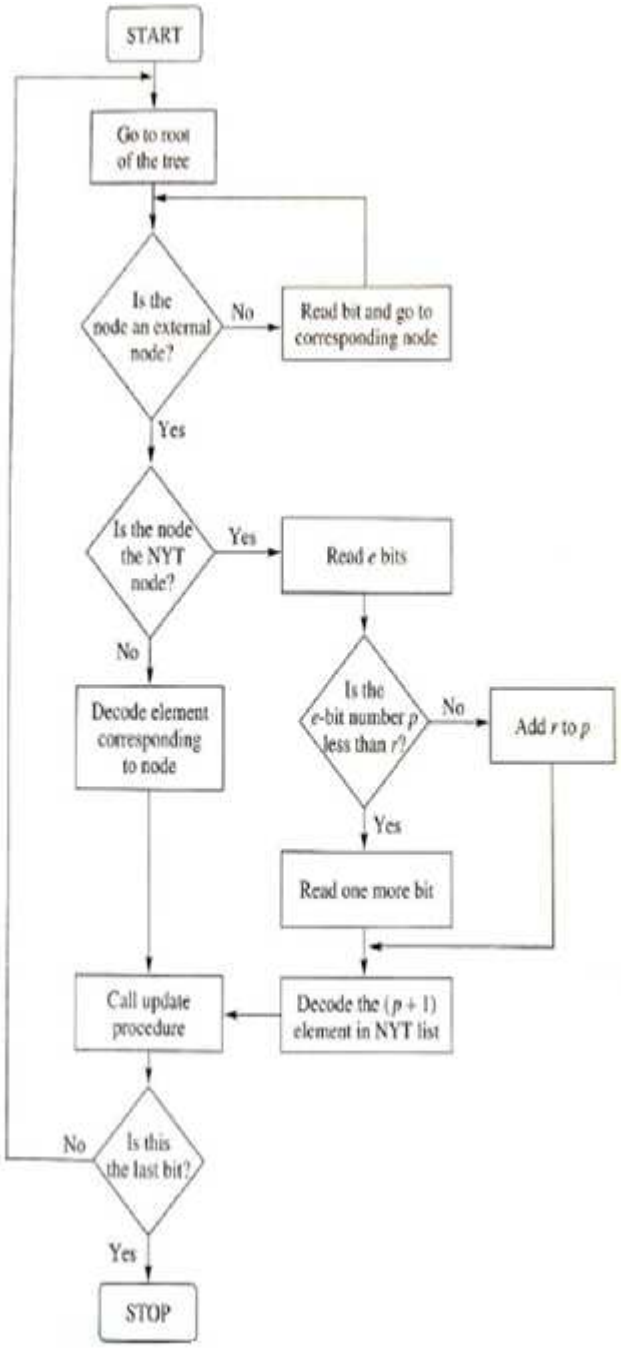
```
/*******************************************************
 * 更新資料結構
 * 參數意義：為第幾個字
 * 呼叫者：FGK_Algorithm_Encode、 FGK_Algorithm_Decode
 *******************************************************/
void FGK_Update(int nI){
    int nq;
    nq = FGK_FindNode(nI);//nq為需要增加weight的external node
    while(nq > 0){
        if((nq < largest[block[nq]]) && (largest[block[parent[(nq+1)/2]]] > nq + 1)){
            exchange(nq,largest[block[nq]]);
            nq = largest[block[nq]];
        }//將nq移到現在block的右邊
        nq = TransferAndAugment(nq);
        nq = parent[(nq+1)/2];
    }
}
```

而以下為 Encode 壓縮檔案的主要程式部分:1. Encode=>從 leaf 往 root 做，再做 update，一個一個運算進來，先知道 short code，在加上樹上的編碼，再整個翻轉傳出。

```
 * FGK Algorithm 的壓縮部份
 * 參數意義：輸入檔名 輸出檔名 何時輸出數據
 * 呼叫者: main
 ******************************************************/
void FGK_Algorithm_Encode(char* cpin,char* cpout,int nTT){
    FILE* fpin;
    FILE* fpout;
    int nI,nIe,nCin;

    int nTemp,nTemp1,nTemp2;
    fpin = fopen(cpin,"rb");
    if(!fpin){printf("無法開啓 %s\n",cpin);return;}
    fpout= fopen(cpout,"wb");
    if(!fpout){printf("無法開啓 %s\n",cpout);return;}

    FGK_Initialize();

    nI = fgetc(fpin);
    nCin = 0;
    while(nI != EOF){
        nIe = map[nI & MASK8];//決定讀到的字是第幾個
        nCin++;
        FGK_EncodeAndTransmit(nIe,fpout);
        if(nCin == nTT){printdata(nCin,FGK_nk,FGK_nCount,FGK_nCBIT);}
        FGK_Update(nIe);
        nI = fgetc(fpin);
    }
    nI = 0;
    FGK_transmit(&nI,fpout);
    fclose(fpin);
    fclose(fpout);
}
```

再來我們將介紹 decode 的流程,解壓縮的部分為跟壓縮的部分為同步過程，

它們之間樹的過程為一一對應，以下為 decode 的流程:



Decode=>從 root 先讀 bit 樹一一往 leaf 找,再把找到的那個點 decode,

而我們所接收到的 code 就是路徑，每 Decode 一個 letter 時， 就須同時

update 我們的樹，所以 decode 我們就把它直接掃過去編碼即可，不用再翻轉。

以下為 Decode 的程式主要部分:

```
/*************************************************************
 * FGK Algorithm 的解壓縮部份
 * 參數意義：輸入檔名 輸出檔名 CB檔名(內寫需解壓縮多少letter)
 * 呼叫者: main
 *************************************************************/
void FGK_Algorithm_Decode(char* cpin,char* cpout){
    int nI,nIe;
    int nj,nN;
    FILE* fpin;
    FILE* fpout;
    fpin = fopen(cpin,"rb");
    if(!fpin){printf("無法開啓 %s\n",cpin);return;}
    fpout= fopen(cpout,"wb");
    if(!fpout){printf("無法開啓 %s\n",cpout);return;}


    FGK_Initialize();
    while(1){
        nj = FGK_ReceiveAndDecode(fpin);
        if(nj == -1){break;}//已經結束
        fputc(rmap[nj],fpout);//輸出
        FGK_Update(nj);
    }
    fclose(fpin);
    fclose(fpout);
}
```

3.3 程式平行處理部分

在我們的實驗程式當中，我們把所切割出來的四個圖片檔案，經由四個同步呼叫的程式，讓四個檔案可以直接由 client 端這邊的 port 直接傳到 server 端的 port 那邊，而四個可以同時呼叫來傳輸，因為網路兩端的 port 四個同時傳輸的執行程式皆不同，在此定為 5001、5002、5003、5004，使的四個傳輸的程式可以同時傳輸且互相不會受到影響，而在 server 端分別收到四個程式傳過來的檔案之後，再開始做高斯處理的流程，以下為平行處理和網路傳輸的程式部分：

(1)四個檔案傳輸部分程式(client 端)：

```
char BUF[2000];
for(int t=1;t<=4;t++){
    sprintf(BUF,"start /B DC_send.exe %d %s %s %s",t,IP[t],Send2Recv_Port[t],Recv2Send_Port[t]);
    if(debug) printf("BUF= %s\n",BUF);
    system(BUF);
    }
```

```
FGK_Algorithm_Encode(Send_Name[t-1],Encd_Name[t-1],0);
char BUF[2000];
sprintf(BUF,"ASIO.exe %s %s %s",argv[2],argv[3],Encd_Name[t-1]);//client送檔案，upload到server，1秒
//   printf("%s\n",BUF);
system(BUF);
```

(2)四個檔案傳輸部分程式(server 端):

```
    char BUF[2000];
    sprintf(BUF,"ASIO.exe %s %s",Send2Recv_Port[ID],"Recv.end");//等case3送檔案
//  printf("%s\n",BUF);
    system(BUF);
//  printf("OK\n");
    //_sleep(2000*(ID-1)+1);
//  printf("FGK_Algorithm_Decode(Recv.end,Work.tif)\n");
    FGK_Algorithm_Decode("Recv.end","Work.tif");
//  printf("FGK_Algorithm_Decode(Recv.end,Work.tif)OK\n");
    My_Image tiff,g_ed;
    //_sleep(200*(ID-1)+1);
//  printf("Li:%d\n",__LINE__);
    tiff.Load_RAW("Work.tif");
//  printf("Li:%d\n",__LINE__);
    g_ed=tiff;
//  printf("Li:%d\n",__LINE__);
    Gen_gauss(13,13);
//  printf("Li:%d\n",__LINE__);
    Gaussian_smoothing(tiff,g_ed);
//  printf("Li:%d\n",__LINE__);
    g_ed.Save_RAW("Gaussian.tif");
//  printf("Li:%d\n",__LINE__);
    //_sleep(200*(ID-1)+1);
//  printf("FGK_Algorithm_Encode(Gaussian.tif,Send.end,0)\n");
    FGK_Algorithm_Encode("Gaussian.tif","Send.end",0);
//  printf("FGK_Algorithm_Encode(Gaussian.tif,Send.end,0)OK\n");
//  printf("Li:%d\n",__LINE__);
    sprintf(BUF,"ASIO.exe %s %s %s",IP[0], Recv2Send_Port[ID],"Send.end");//clinet，upload檔案到case3
//  printf("Li:%d\n",__LINE__);
//    printf("%s\n",BUF);
    system(BUF);


    sprintf(BUF,"ASIO.exe %s %s",argv[4],Decd_Name[t-1]);//serever架構，所以會等case5回送檔案 ，總還回來34秒
//    printf("%s\n",BUF);
    system(BUF);
    FGK_Algorithm_Decode(Decd_Name[t-1],Recv_Name[t-1]);
```

3.4 檔案一切四和四合一部分

原本的檔案為較大的一張原始的圖檔案，大小為 4.10MB 的圖片，我們因為考量到原本的檔案過大會造成傳輸時間過長的問題，所以先把原始的檔案先切割成四張，而切割的時候，會造成檔案不完整，所以把另一邊的圖的 50 個 pixel 值覆蓋到另一端的地方，另外邊緣的兩邊就補 0 來做影像的調整，例如形成的圖形如下:

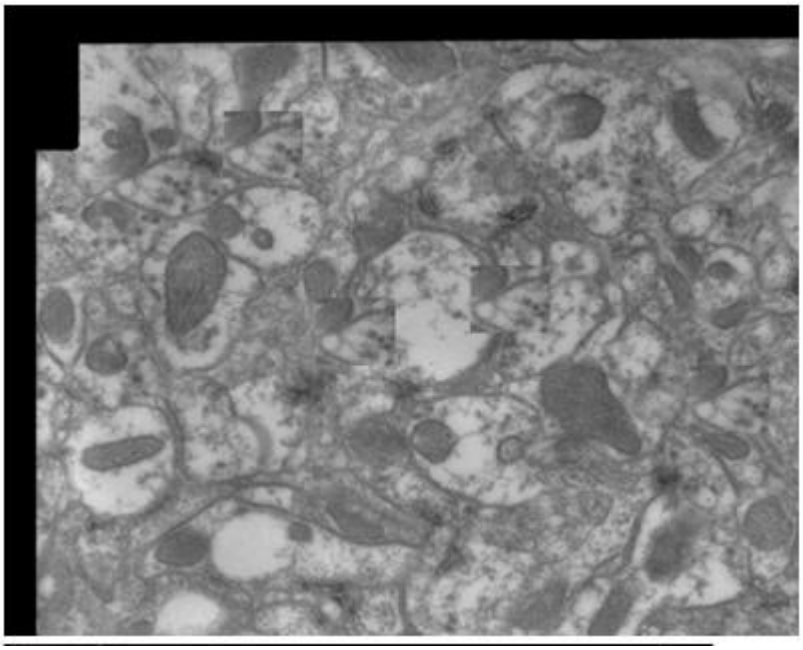


圖 6.切割完後左上方的圖片

這是一張圖片左上角切割的圖形的部分，即把右邊和下方的 50pixel 的值來補足切割的地方，而左邊和上方則補 0 來取代。

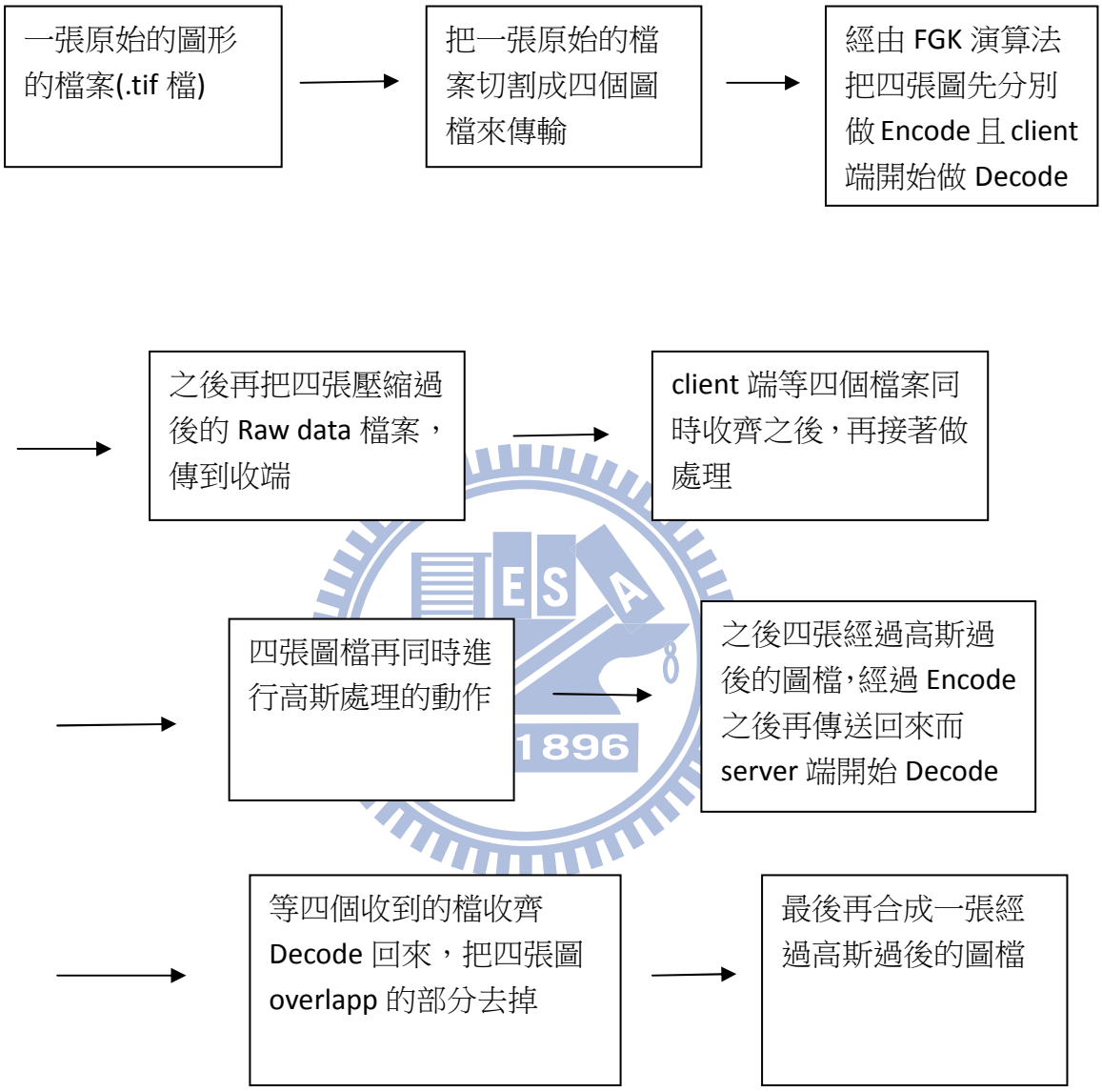
而最後當我們收到四個經過高斯處理之後的圖片同時傳送回來之後，再把之前 overlapp 的部分把它去掉，之後再把四張圖片合成起來，這樣就可以達到我們最後所需要的結果，比單獨一張傳過去收端，經過高斯處理後再傳送回來，省去大部份的時間。

3.5 網路傳輸部分

在網路傳輸的部分，我們是採取 TCP 的傳輸模式，我們把它寫成像是 FTP 傳輸的功能，分為兩個部份， client 端即為我們送端這邊所呼叫的程式，它把 Raw data 的檔案讀進來之後，之後再一個一個 byte 寫出去 network，而另外一邊 server 端則是收端這邊所呼叫的程式，把它對應的 port 所傳輸過來的 data，從 network 傳輸過來的 Raw data 資料讀進來之後，再把一一對應的 byte Decode 回來，如此即為基本的傳輸的原理實做部分，目前可以用一台本機電腦，和四台其他端的電腦來做實做此次實驗所需的圖片，也可以一台本機電腦，和兩台其他端電腦來做測試。

3.6 即時動態壓縮影像系統總覽

實驗流程總覽:



第四章　實驗結果分析

4.1 實驗簡介

我們把第三章所提到的整個程式流程，來把它寫成程式來做實際的測試，當中會有圖檔切割部分、FGK 演算法的實作部分、和網路傳輸的部分、以及經過高斯處理的部分，最後合成為一張單張的圖片。

在本章節中的資料則是採用國家同步輻射中心所提供的資料(.tif)檔的資料，與另外兩張國家同步輻射中心所另外提供的照片兩張(.tif)，而自己從網路上找一張較大的圖來做測試，來做多張圖的實際檢測與模擬。

4.2 實驗環境

Client 端電腦:

1. CPU:Pentium 4 cpu 3.20 GHz
2. RAM:2.00 GB
3. OS system：Microsoft Windows XP Professional Release mode
4. . Develoment environment: DevC++

Server 端電腦:

我們使用實驗室的另外其他 2~4 台電腦當作 server 端，來實做出我們所需的 1 to 2 ，和 1 to 4 的功能，同樣也是在 Windows 介面下來做處理。

4.3 實驗結果

1. 國家同步輻射中心圖形檔案傳輸結果:

此模擬的原始影像大小為 2349 * 1823 ，4.10MB 的大小，如下圖 7，它是屬於(.tif)的檔案格式形態，而我們的第一張圖對此圖做切割和傳輸的分析。

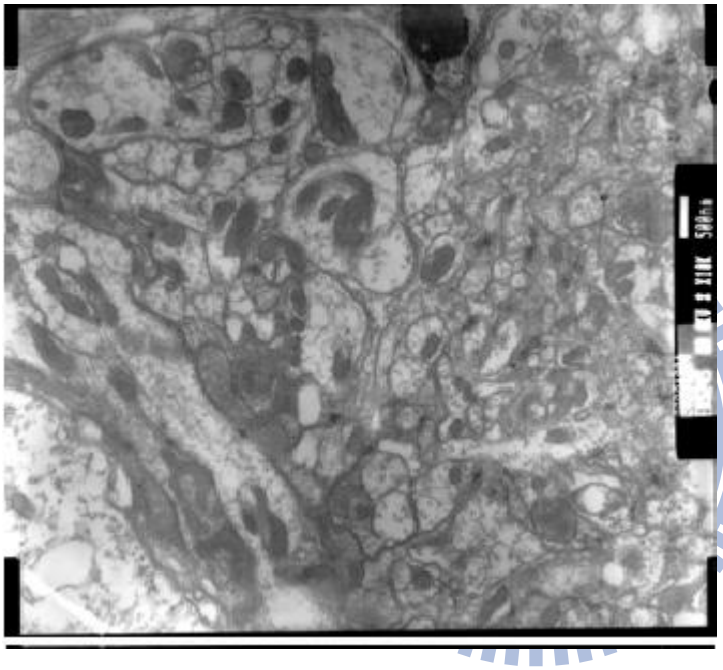


圖 7.國家同步輻射中心所提供之原始資料

我們一開始先把原始的一張圖片切割成四張圖片，切割後的四張圖片為等大小的圖片，大小四張皆為 3.49MB

之後，我們再經由 FGK 演算法壓縮，一個一個 byte 把 Raw data 的資料傳到網路上，之後 server 端四個 port 會把對應的 port 一個一個 byte 來接收 raw data 的資料，再把收到的資料一個一個 byte 解壓縮回來，等四個切割的檔案收

齊之後，再分別同時進行高斯處理，

而在處理完高斯處理之後的圖片後，我們再把四張經過高斯處理的圖片一個一個 byte 壓縮成四份 Raw data 的檔案，再從 server 端的 port 反傳回對應的 client 端，一個一個 byte 解壓縮回來，而等收齊四份 Raw data 的檔案之後，，之後再把四份圖檔經過把 overlapp 部分去掉後，四張合併成一張經過高斯處理過後的圖檔，圖 8 為最後送端得到的最後回傳回來的圖檔。

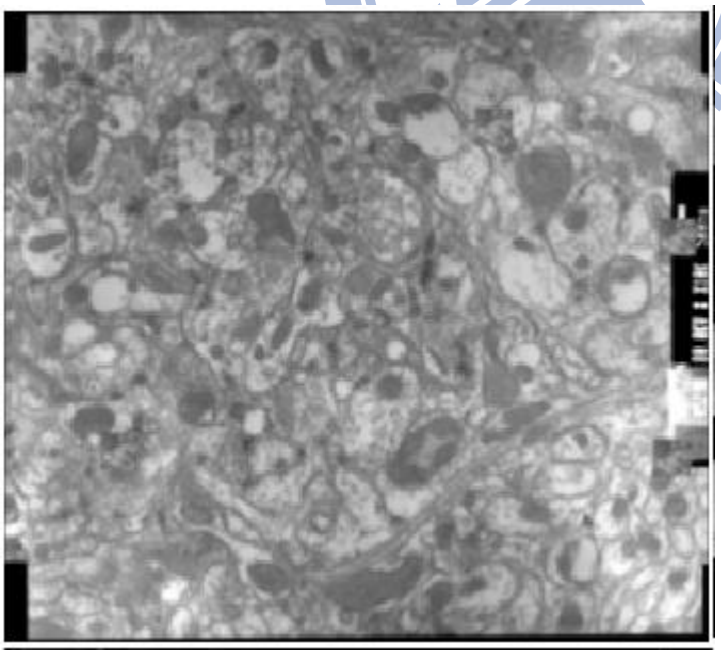


圖 8. 最後經過高斯處理回傳回來的圖檔

第二筆原始檔案的圖檔，檔案大小為 2349*1823，圖 9 為原始的資料檔案，而圖 10 為我們實驗過後所產生的圖檔:

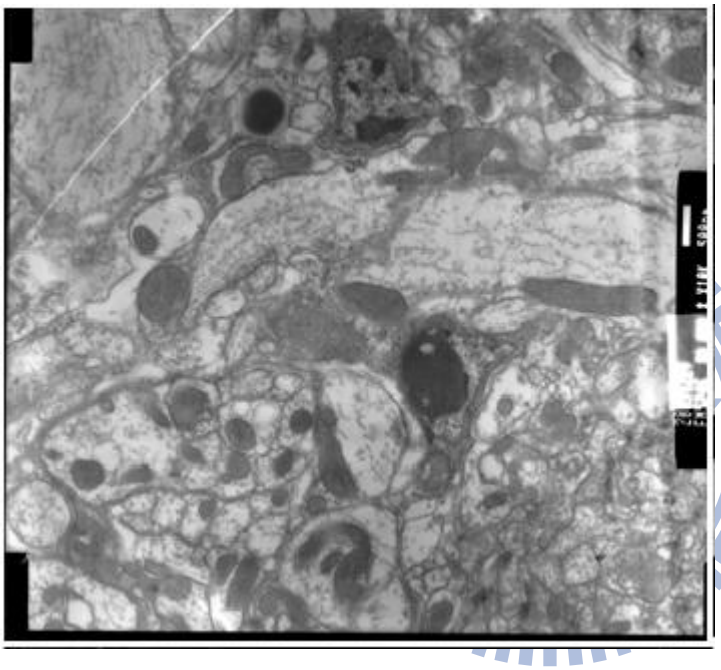


圖 9. 第二筆原始的圖檔

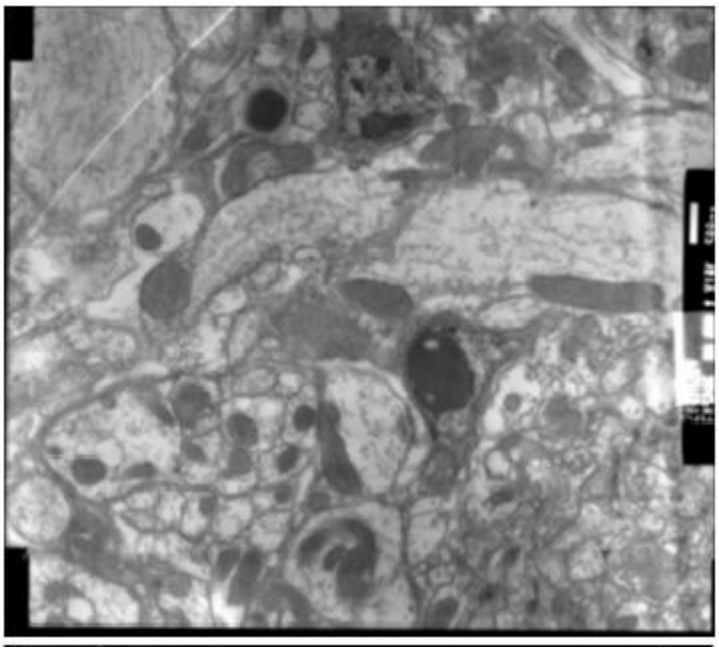


圖 10.經過實驗過後產生的圖檔

第三筆原始檔案的圖檔，檔案大小為 2349*1823，圖 11 為原始的資料檔案，而圖 12 為我們實驗過後所產生的圖檔:

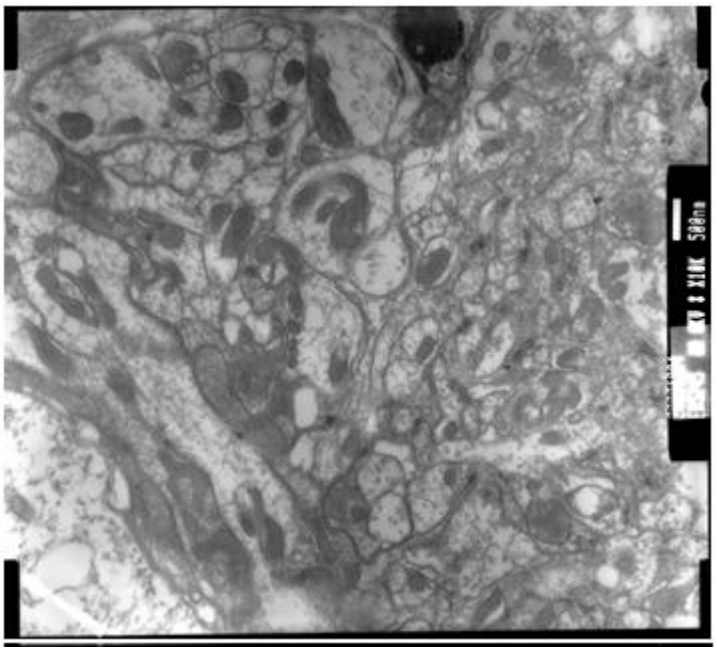


圖 11. 第三筆原始的圖檔

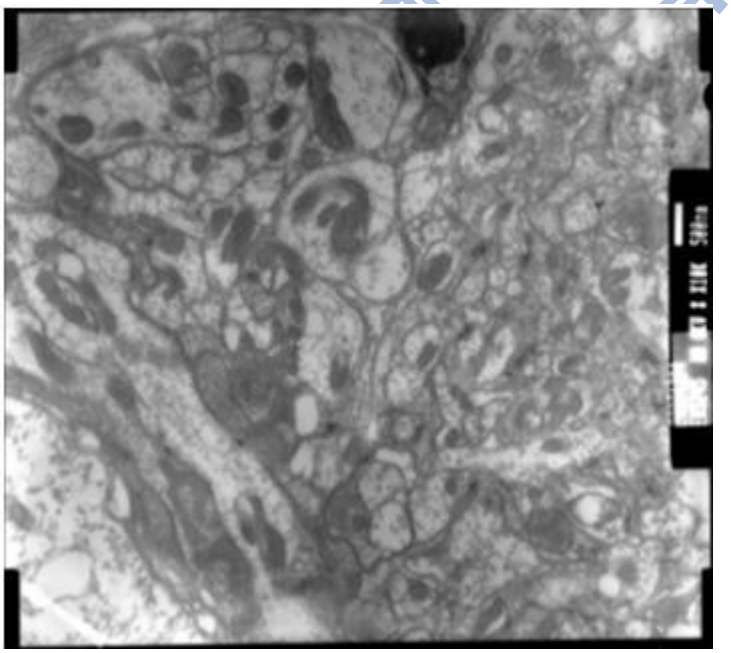


圖 12. 經過實驗過後產生的圖檔

我們最後把系統核心的程式所需的時間做一個整理，總共測量了單一台電腦所做高斯處理所需的時間，和 1 對 2 台電腦所需的時間，以及 1 對 4 台電腦所需的時間，三個時間和多張所測試的圖片做個整理表格如下:

圖片大小	4MB	49MB	110MB	196MB	306MB
單一運算(s)	180	808	2113	3670	5793
Computer1 Gaussian(s)	11	42	117	162	254
Computer2 Gaussian(s)	38	123	233	392	609
Computer3 Gaussian(s)	20	103	112	208	317

Computer4 Gaussian(s)	13	97	167	371	471
Total time (s)	264	773	2650	3670	5793

我們得到了以下的結論，因為我們放大圖片和做 Gaussian，在網路的傳輸為固定量的傳輸，而做 Gaussian 其實也是 O(nk)的成長，K 為 Gaussian 的 kernel，n 為圖片的 pixel 數，所以我們得到了如果放大圖片並不會得到較好的 speedup 的結論。

我們想出了兩種解決此種情況的方法，第一種為在我們做 operation 的動作時(即做 Guassian 時)，加大它的 delay 的時間，而我們試過幾個 delay 的值後，，加大 Delay 值後調整到的約可以達到以下的約 4 倍左右的 speedup。

	第一張圖	第二張圖	第三張圖	第四張圖
圖片大小	3104*4672 41.5MB	2349*1823 4.1MB	2349*1823 4.1MB	2349*1823 4.1MB

單一運算(s)	5921	1439	1278	1393
1 to 2 運算(s)	1650	373	352	329
1 to 2 的 speedup	3.58	3.86	3.63	4.23
1 to 4 運算(s)	1296	326	242	267
1 to 4 的 speedup	4.57	4.41	5.28	5.22

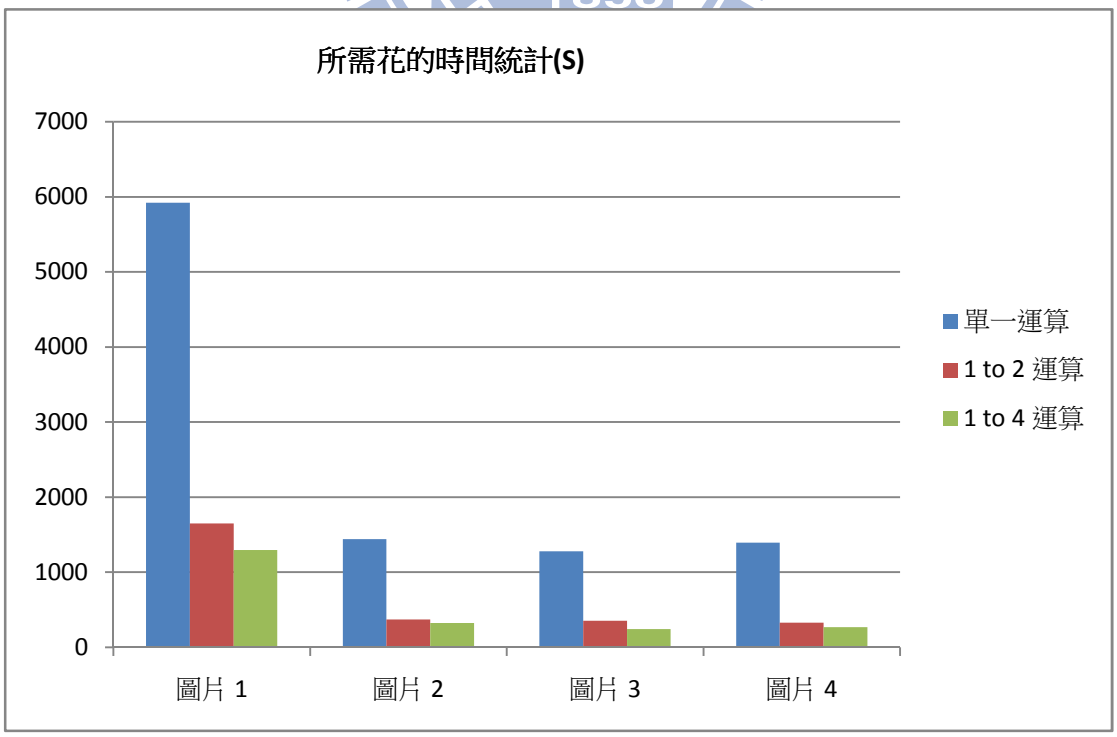


圖 13. 上方表格三種執行時間之圖表整理

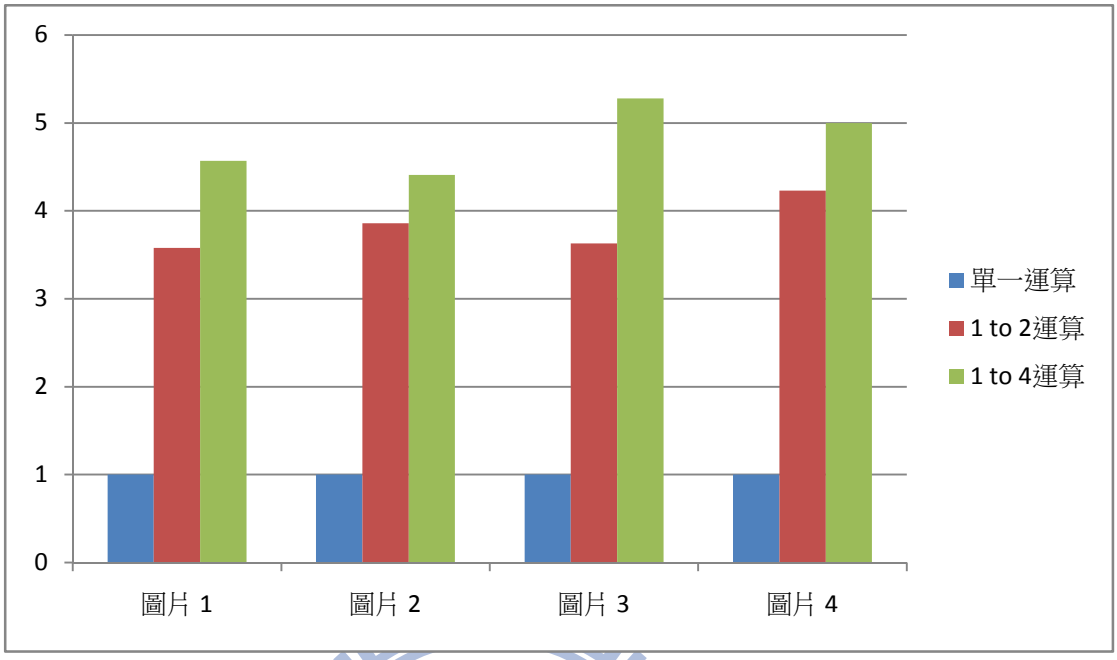


圖 14. 上方表格三種運算的 speedup 之圖表整理

我們由以上的實驗中可發現由我們所提出的方法，不管是 1 to 2 的系統架構，或是 1 to 4 的系統架構，皆能有效率的提升我們原本單一運算在單一電腦所需的時間，而做出來的測試中，最少增快的 1 to 2 speedup 也可達到 3.58 倍，而最快的 1 to 4 speedup 的部分可以達到 5.28 倍，這樣的做法表示了的確有實質上速度的幫助，讓我們在處理大檔案的時候可以做更快速的處理。

第二種測試的方法為一次傳送一個檔案來做時間的檢驗，我們先把檔案整個先壓縮完後，再分批傳送出去，如此測試也有相當的效果，以下為測試的相關數據:

圖片大小	4MB	49MB	110MB	196MB	306MB
Computer1 Gaussian(s)	11	42	93	162	253
Computer2 Gaussian(s)	20	108	230	407	712
Computer3 Gaussian(s)	13	50	113	198	308
Computer4 Gaussian(s)	20	76	169	299	464
Total time (s)	47	180	422	843	1399
單一運算(s)	180	808	2113	3670	5793
增加的 speedup	3.83	4.49	5.00	4.35	4.14

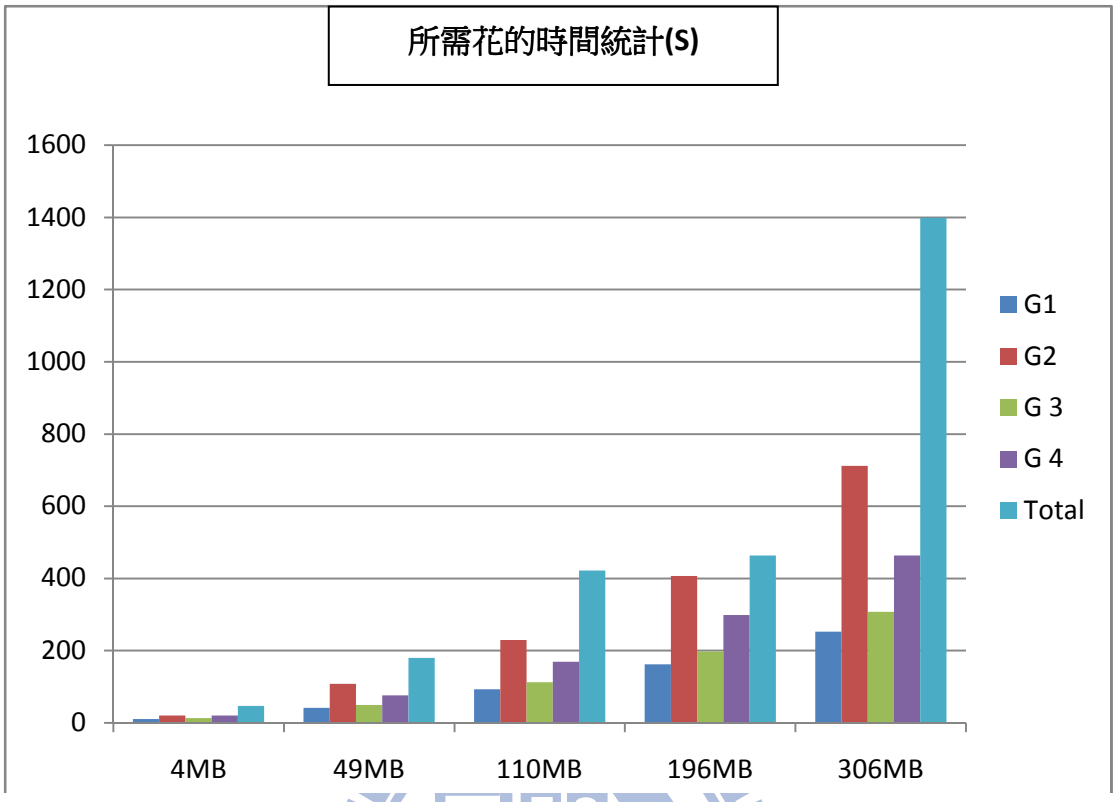


圖 15. 上方表格四個 Recv 端 Gaussian 和 Total 執行時間之圖表整理

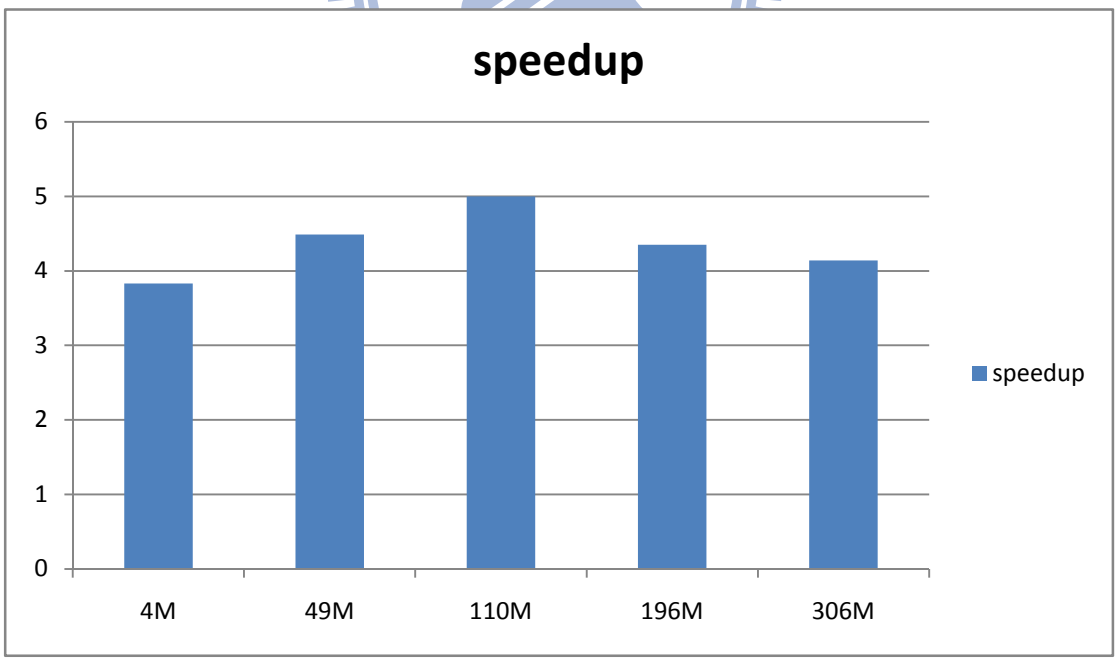


圖 16. 上方表格整批檔案傳輸運算的 speedup 之圖表整理

我們由以上整張檔案壓縮後再傳送的處理程序中，以 1 to 4 的架構下，可以得到以下的相關的資訊，在處理從小檔案 4MB 的圖片一直到大檔案的圖片中 306MB，我們均可以獲得 3~5 倍間左右的 speedup 對於單一運算而言，這樣的做法表示了的確有實質上速度的幫助，讓我們在處理大檔案的時候可以做與之前 FGK 動態壓縮演算法做時間上的處理的對比。

第五章　結論與未來展望

在本篇論文中，我們運用了一些相關的演算法和網路傳輸的技術，應用在平行處理電腦斷層掃瞄的相關圖片上，我們以 FGK 演算法為基礎的核心架構，使用多個 process 同時傳輸來加速影像切割和合併所需耗費的時間，節省原本使用 Huffman 演算法和只靠單一 process 運算長久時間上等待。

使用別的演算法來代替 FGK 是否為可能加速時間上的效率，這是本篇論文未做到的事情，或是切割更多塊，讓更多 process 來同時執行傳輸，是否可能加速時間上的效率，目前這方面有相當多的研究文獻皆在討論該方面的問題，是未來可以值得繼續相關研究的議題。

參考文獻

[1] R. G. GALLAGER. Variations on a theme by Huffman, IEEE Trans. Inform. Theory IT-24 (1978). 668-674.

[2] D. E. KNUTH, Solution to problem E2307, Amer. Math. Monthly 79 (1972), 773-774.

[3] KNUTH, D. E. Dynamic Huffman coding. J. Algorithms 6 (1985), 163-180.

[4] VIITER, J. S. Dynamic Huffman Coding. ACM Trans. Math. Sojlw. Submitted 1986.

[5] VITTER, J. S. Design and analysis of dynamic Huffman codes. J. ACM 34, 4 (Oct. 1987),825-845.

[6] BENTLEY, J. L., SLEATOR, D. D., TARJAN, R. E., AND WEI, V. K. A locally adaptive data compression scheme. Commun. ACM 29,4 (Apr. 1986), 320-330.

[7] ELIAS, P. Interval and recency-rank source coding: Two online adaptive variable-length schemes.IEEE Trans. InJ Theory. To be published.

[8] FALLER, N. An adaptive system for data compression. In Record ofthe 7th Asilomar Conference on Circuits, Systems, and Computers. 1913, pp. 593-591

[9]VIITER, J. S., AND CHEN, W. C. Design and Analysis of Coalesced Hashing. Oxford UniversityPress, New York, 1987.

[10]D. E. KNUTH, Solution to problem E2307, Amer. Math. Monthly 79 (1972), 773-774.

[11]JEFFREY SCOTT VITTER “ALGORITHM 673 Dynamic Huffman Coding”

ACM Transactions on Mathematical Software, Vol. 15, No. 2, June 1989, Pages 158-167

[12] T.C. Bell , J.G. Cleary , and I.H. Witten , Text Compression , NJ: Prentice Hall , 1990

[13] Ross N. Williams “Adaptive Data Compression”

[14] http://en.wikipedia.org/wiki/Speedup

[15] https://computing.llnl.gov/tutorials/parallel_comp/