利用叢集架構整合個人電腦運算能力

林作俊[1]　劉思源[2]

1、國立宜蘭技術學院電子工程系副教授
2、國立宜蘭技術學院電子工程系學生

摘 要

與人類生活福祉息息相關問題的解決，有許多需要藉助高速運算平台。電腦叢集以其具有高速運算、低成本之特性，受到學界及工商業的青睞。除此之外，其本質上亦具有很好的擴充性及容錯性，因此它在許多平行與分散式處理的應用也一直受到重視。本篇說明宜蘭技術學院電子系之個人電腦叢集架構之實現，此平行處理環境目前是以不同規格的個人電腦建構成異質運算環境，我們除了在此電腦叢集發展平行處理的技術，亦將它做為平行程式設計的教學平台。在此篇報告中，為了展現叢集運算能力整合之成果，我們選擇了三種不同特性的問題：(1)π值之計算(2)數字排序和(3)最佳通訊地點之選擇，以進行實測、數據分析與檢討，實驗結果顯示電腦叢集系統，針對這三類問題都有相當好的加速比表現。

關鍵詞：電腦叢集，平行處理，分散式處理，MPI。

Integrating the Computation Capability of Personal Computers Using Cluster Architecture

Cho-Chin Lin[1] Si-Yuan Liu[2]

1. Associate Professor, Department of Electronic Engineering, National IIan Institute of Technology
2. College Student, Department of Electronic Engineering, National IIan Institute of Technology

Abstract

Many of the applications concerning the issues of human welfare need a large volume of computations. Computer clusters can provide high performance computing environment with limited budget. Thus, many people employ computer clusters to suit their computational requirements. Its high scalability and fault tolerance is suitable for many parallel or distributed processing applications. In this paper, our heterogeneous cluster which is built for academic purpose is reported, and three problems of various characteristics, (1) π calculation (2) column sort and (3) the selection of optimal communication location, have been solved to demonstrate the usefulness of our computer cluster. The experimental result is given. It is shown that our computer cluster achieves good speedup for the three problems.

Keywords: computer cluster, parallel processing, distributed processing, MPI

一、前言

與人類生活福祉息息相關的問題，有許多可以藉助計算機之運算而獲得寶貴的資料或答案，例如：長期天氣預測、人類疾病藥物之研發、能源之開發等等。此類的運算資料量相當龐大，因此對於運算平台速度的要求也特別嚴苛。以長期天氣預測來說，若大氣資料的採樣密度為每五公里一個單位，則計算機的運算速度必須大於 20Tflops，以便有效且精確的預測天氣[1]，1 Tflops 相當於每秒執行 10^{12} 次浮點運算。另一個例子就是高速運算在生命科學上的應用，美國 Celera 生物科技公司從事基因定序時，某個定序步驟(whole-genome assembly)需要 20,000 個 CPU 小時計算[2]，為縮短計算時間，該公司利用速度為 1.5 Tflops 的計算機搭配了 100Tbytes 的儲存空間從事運算，並計劃未來做蛋白質結構分析時將計算機的運算速度擴大為 100Tflops，以符合運算的需求。由以上的例子，我們得知許多重要的應用必須依賴高速運算平台。根據目前半導體的技術，近幾年來，微處理器運算速度每年增加 50%至 100%，記憶體資料容量的密度每 3 年增加四倍[3]，但根據物理定律，運算速度將受限於光速，使得微處理器的運算速度最終將達到極速，單一 CPU 或少數 CPU 所建構的超級電腦勢必無法滿足高速運算的需求，因此使用大規模平行處理(Massively Parallel Processing)是無法逃避的現實。目前許多公司積極籌建超級電腦以滿足此類運算上的需求，例如 IBM 計劃建造運算速度達 3.88Tflops 的「太平洋藍」以模擬核子試爆[4]，Intel 計劃建造運算速度達 13.6Tflops 的「TeraGrid」以從事生物細胞模擬及藥物開發[5]，Paracel 亦開發了一部擁有 27648 個 CPU 的超級電腦用以執行基因序列的分析[6]。

在全新的運算架構(例如：quantum computation[7])成熟之前，用平行處理技術解決複雜問題似乎是唯一的方向，實現大規模平行處理的方式有許多種，叢集的架構在效能價格比的考量下是一個很不錯的選擇，建造電腦叢集架構的基本元件是一般市面上可獲的電腦及網路元件，為了讓這些電腦可以從事高速的運算，利用系統軟體將多部電腦主機之運算能力整合，任何硬體、軟體與網路零件，只要是 COTS（commodity-off-the-shelf），都符合叢集系統元件的定義。因為所組成的元件都是大量生產的元件，因此價格會比較便宜，所發揮的效能將可帶來更多的利益與價值。

由於電腦叢集架構可用來實現大規模平行處理，且價格便宜，因此我們計劃自己建造用於教學與研究的電腦叢集。我們的電腦叢集是採用異質的模組，也就是說每部電腦的硬體規格不盡相同。異質模組較符合實際擴充的情況，因為一般要新加入叢集系統的電腦模組會比原先已加入叢集的電腦模組來的更高階。但因為各個模組在設計之初，並非專為高速運算設計的，整合後的運算能力，極有可能無法達到預期的效果，必須要靠軟體設計及平行處理的技術以提昇其整體的效能，目前有許多相關的研究正進行中[8,9,10,11,12]，對於未來超高速電腦發展及應用有興趣的讀者，可以參閱[13]。本篇說明宜蘭技術學院電子系之個人電腦叢集架構之實現，此平行處理環境目前是以不同規格的個人電腦架構成異質運算環境，我們亦在此叢集架構上規劃設計平行處理程式，並在此叢集系統上進行實測、數據分析與檢討，實做的題目包括了(1)π值之計算(2)數字排序和(3)最佳通訊地點之選擇，都獲得相當好的加速比。

二、背景

叢集架構的構想源起於 1994 年夏季，美國太空總署 CESDIS(Center for Excellence in Space Data and Information Science)研究中心為了進行地球與太空科學研究計畫，嘗試使用低廉而易於取得的電腦相關零件來組裝可支援平行計算的電腦系統，以因應該計畫必須處理的大量資訊。它的組裝元件是將 16 部 CPU 為 Intel 100MHz DX4 的個人電腦，由 10Mbps Ethernet 整合成一個叢集系統，並取名為 Beowulf。1996 年九月，美國的 Los Alamos 國家實驗室建構了另一部電腦叢集系統，取名為 Loki。它是由 16 部 CPU 為 Intel Pentium Pro 200 的電腦所整合，網路採用 100Mbps Fast Ethernet，作業系統為 Linux，資料傳輸介面採用 MPI（Message Passing Interface）通信程式庫。約略在同一時期，美國的噴射推進實驗室與加州理工學院也建置了一個極類似的叢集用於太空任務，並將其取名為 Hyglac。類似 Beowulf 的叢集架構之所以能成功與風行，是因為它很適時地把握住了微處理器及相關電腦科技的發展趨勢與潮流[14]，過去十年間，COTS 產業發展蓬勃快速，連作業系統、系統軟體、工具軟體、資料傳輸軟體都可以在網路上以免費或低廉的價格下載；另一方面，高速計算的需求與日俱增，平行計算與處理的技術日趨成熟。電腦叢集的優點包括：不受單一廠商牽制、具選取及更新組件的彈性、可因應運算需求而拓展其規模、可隨科技進展，快速調整零組件、採用一般之冷卻系統。當然，最重要的是個人電腦叢集在高速計算領域已被證實有很優越的效能價格比。舉例來說，2000 年 4 月由 Thomas Hauser 所建置的「Kentucky Linux Athlon Testbed 2」，其費用約需美金四萬元。當時它測試效能評估程式，得到的最高速率是 64Gflops，這個成績使得 KLAT2 於 2000 年 6 月登上世界五百大電腦系統名單上的第 197 名[15]，這樣的效能價格比約為大廠商製造的超級平行電腦的十五倍。

“叢集”顧名思義是許多個運算小單位的組合，此運算小單元可以是功能簡單的個人電腦，也可以是功能完備的工作站。電腦叢集可以透過單一系統形象（Single System Image）的設定，達到資源共享，使用者無須知道其所用資源所在位置，也無須指定使用的運算單元，完全由叢集的分散式作業系統來掌控，從整體觀點來看，外界或客端電腦視電腦叢集為一部超級電腦或高效能伺服器，客端電腦的運算需求，可透過網路輕易的在高效能的電腦叢集端執行。若這些電腦的軟硬體規格相同，則此叢集稱為同質性叢集，若軟硬體規格不同，則此叢集稱為異質性叢集。在電腦叢集系統中，這些運算小單位透過網路的整合，複雜的工作藉由平行或分散式處理的技術，將該工作切割成許多子工作，再依子工作的特性指定給不同的電腦執行，藉由叢集內所有電腦模組的分工合作，在最短的時間內完成該工作。總而言之，電腦叢集的主要優勢有：

(1) 透過網路的整合，可輕易提昇叢集的整體運算能量 ·

(2) 可透過工作量的平均分配，爭取運算時效。

(3) 記憶體容量，可隨叢集的數量而獲得擴充。

(4) 效能價格比，超越超級電腦，經濟效益好。

我們此篇所談之叢集，其電腦分佈所涵蓋的範圍是屬於區域性的，大部分皆在 100 平方米的範圍內，另有一類高複雜度的叢集，我們稱為 GRID[16]，其電腦分佈的範圍涵蓋整個地球。電腦叢集應用之

範疇，可取代一般電腦伺服器主機之工作，服務廣大的客戶端電腦，甚至在尖端科技所需要的龐大運算工作，如天文、醫學、武器方面等等許多科學領域，皆有相當好的運用空間或發展潛力，而近程較易實現的應用為工程領域中大型題目解題之計算。

三、宜技叢集系統

我們的叢集系統以四部單一 CPU 的個人電腦所組成，分別命名為 Tree1, Tree2, Tree3 及 Tree4，作業系統採用 Linux Redhat 6.1 版，核心為 2.2.12，主機間以 100 Mbps Fast Ethernet 整合，並共享 100M 的通訊頻寬，叢集名稱為「森林」，架構如圖 1-a 所示，實物如圖 1-b，硬體規格如圖 1-c 所列。各個電腦之間的通信採用 MPI 的標準[17]，因此 Fortran、C、C++ 等語言所撰寫的程式可以利用呼叫 MPI 程式庫的方式來通信，目前最新的 MPI 標準是屬於 MPI2.0 版。MPI 提供了一百多個通信指令，程式設計師可以依照本身的需要選用某些較有效率的通信指令來完成複雜的工作，但值得注意的是，若效率不是特別的要求，一般來說，六個指令已經夠用。除了在 Linux 上的 MPI，目前也有廠商提供 NT Server 的 MPI 程式庫。為了讓各個電腦間在執行程式時能輕鬆的讀取並執行程式，我們啟動了 Linux 之 NFS (Network File System)的功能，NFS 是由昇陽電腦公司(Sun Microsystems)為 SunOS 發展出來的分散式檔案系統，可讓使用者透過連接的網路，將其它電腦同意對外分享的檔案目錄，掛載到目前所使用之電腦的檔案系統下，使用者在存取這些檔案或目錄時，感覺就如同是儲存在本機上一樣。由於電腦叢集系統中每一部主機都必須在相同的使用者帳號及目錄下放置相同的執行檔，才能順利同時啟動並執行，利用 NFS 可以省去複製檔案的工作，使得每一部電腦皆掛載某主機分享出來的工作目錄及其下的所有檔案。因 NFS 是 Linux OS 預設安裝的功能之一，只要將其啟動為系統常駐服務程式即可。另外，我們亦啟動 NIS (Network Information System)，以使得叢集系統中所有電腦模組的系統設定檔有一致的參數設定與規劃。

四、測試程式

我們選擇三個性質非常不同的問題以展現電腦叢集的運算功能，第一個問題為π值之計算，此為相當容易平行化的運算，經由審視計算π值所用的時間，我們可以概估兩部電腦間的運算速度比及運算所需的通信時間。第二個例子是數字排序，電腦之間的通信較為複雜，此問題最主要是說明解決同一問題使用不同平行演算法將需要不同的時間，採用某些非最佳化的演算法執行運算時，於量測加速比時會產生超線性加速比的現象(Superlinear Speedup)。 第三個例子是一個應用的程式，用於搜尋月球表面之最佳通訊地點，我們用此題來說明如何在電腦叢集上設計演算法，以解決高複雜度的問題。

(一) π值之計算

它的演算法是以利用函數 $y=\sqrt{1-x^2}$ 在 $0 \le x \le 1$ 以二維空間所圍成的面積之和來求得。計算的

方式是將 0 到 1 的 x 軸分割成 N 個區段$[x_0,x_1]$，$[x_1,x_2]$，… $[x_i,x_{i+1}]$ …$[x_{N-1},x_N]$，並定義由點 $x_i x_{i+1}\sqrt{1-x_i^2}\sqrt{1-x_{i+1}^2}$ 圍成的區間面積為 A_i，則 $\pi \approx 4\times\sum_{i=0}^{N-1} A_i$，$N$ 值越大，計算所得之π值越準確，若要用 W 個電腦來計算π值，為使計算的時間縮短，則必須使每部電腦執行的工作量約略相等，也就是每個電腦約負責 N/W 個區間。此演算法不需要太多通信，只需在執行前讓每部電腦知道它所負責的區間及在執行的最後將各部電腦計算的結果送至一部指定的電腦以利最後的加總，所以 W 部電腦在此步驟的總運算量為 $O(N)$。由於此問題平行化所用的技術相當直接，我們最主要是利用此題獲得電腦叢集的系統參數，並將觀察所得的系統參數與電腦叢集的實際配備比較，因此我們只針對擁有一部及兩部電腦模組的叢集取得其運算的數據。測試的過程中，以 Tree4 為啟動運算之電腦模組並在整個運算過程中參與運算，若有第二部電腦模組需要參與運算，則以 Tree1 為該運算模組。

運算之數據顯示於表 1。從該表中可以得知當 N 較小時，單一電腦模組之叢集較兩部電腦模組之叢集來的快，但分割量漸多，到達相當大的一個數量以後，多部主機的平行計算優勢就可明顯從速度上看出了。這是因為兩部電腦模組之叢集在 N 較小時，通信時間佔了整個運算時間的大部份。我們利用該現象求出在整個運算過程中，兩部電腦模組的通信時間約略佔了 290μs(10^{-6}秒)。當 N 較大時，通信時間在整個運算時間中幾乎可以被忽略，利用此特性，我們可以得知 Tree4 的計算速度約為 Tree1 的 1.14 倍，此觀察結果與我們所用的硬體規格吻合。

(二) 數字排序

我們利用 column sort 演算法[18]來做數字排序，column sort 是一種排序的平行演算法。一般而言，電腦間的資料交換在叢集架構經常需要軟體的介入，因此相當耗費時間，而 column sort 在整個排序的過程中，只需四次的通信步驟，因此它特別適合用於電腦叢集的架構上，其中兩次的通信步驟，每部電腦均需將其運算結果分配到其它的電腦，此類通信的方式稱做多對多(all to all)通信，是平行處理中最複雜的一種通信型態。Column sort 在每個通信步驟之間所執行的計算也是排序的運算，但此排序是每部電腦模組針對各自分配到的數字執行排序，我們將它稱作區域排序。總括來說，column sort 完成排序的方式是將透過數次的區域排序並參雜數次的資料交換而完成。若系統中有 W 部電腦參與排序，則我們將這些待排序的數字劃分成 W 組，一部電腦負責一組數字之區域排序。在此篇報告中，區域排序的演算法，我們分別採用 bubble sort 與 quick sort[19]，以觀察並比較使用不同區域排序時的整體效能。我們在測試的過程中將所有以亂數產生的數字全部集中在 Tree4，並以 scatter 指令分配資料到其它電腦，運算後其結果以 gather 的指令將結果集中至 Tree4。然而我們所量測的時間，不含 scatter 及 gather 指令執行的時間。量測時，分別啟動叢集一至四部電腦模組進行測試，測試時採用多次採樣。

採用 bubble sort 為區域排序時，時間與資料量之關係顯示於圖 2-a，採用 quick sort 為區域排序時，時間與資料量之關係於圖 2-b。我們發現 bubble sort 的區域排序策略，其運算所需時間遞增速率遠高於資料增加之速率，因為 bubble sort 是一個時間複雜度為 $O(N^2)$ 的演算法，其中 N 為排序

的資料量，因此造成時間-資料量之曲線呈非線性上昇。若採用 quick sort 為區域排序時，其運算所需時間遞增速率約等於資料遞增速率，雖然 quick sort 的平均時間複雜度為 $O(N\log N)$ 的演算法，但因所測的資料量差距不大，$\log N$ 的影響可以忽略，因此造成相對之上昇曲線幾乎呈線性之關係。而由圖 2-a 與圖 2-b 相較之下，可以看出於區域排序使用 quick sort 演算法明顯優於將 bubble sort 演算法使用於區域排序。同樣的，利用多部電腦模組的叢集來加速運算，處理的資料量越大，速度提昇得越多。我們進一步計算電腦叢集系統對於此 column sort 的效能加速比，加速比 Speedup(W) 之定義為 1 部電腦解決某個問題所需的時間除以具有 W 部電腦模組的叢集在相同的資料量下解決相同問題所需的時間。使用不同區域排序在不同的資料量下所得的加速比顯示於圖 3，其中使用 bubble sort 時，其加速比之值大於電腦個數 W ，顯現出超線性加速比的現象。一般來說，加速比之值應不大於電腦個數 W，造成超線性加速比的原因有可能來自硬體(例如：記憶體或快取記憶體總容量隨著主機模組數增加，因此資料存取的時間會減少)或演算法(例如：採用次佳演算法[20]，一般來說，此類的演算法中，其多餘的工作量(redundant work)會隨著資料量而增加，因此每部主機所處理的資料量減少時，執行的時間會減少許多)，而我們於區域排序採用 bubble sort 時資料量不是很大，因此記憶體的影響應不明顯，會造成的超線性加速比之現象，研判其原因可能 bubble sort 不是排序的最佳演算法所致。

(三) 最佳通訊地點之選擇

最佳通訊點的問題定義於[20]，我們將此一問題簡略描述如下，某次月球任務中，有一太空船登陸月球從事蒐集地表環境資料的任務，資料蒐集的範圍涵蓋降落區數十平方公里以內的區域。由於地表崎嶇不平，為了蒐集到的資料能涵蓋各處死角，降落後發射了許多小型機器人協助蒐集，蒐集後，機器人不再回收。每個機器人負責某一特定地區的資料蒐集，並將所蒐集到的資料藉由無線通訊傳回太空船，由於地表崎嶇不平，有高山有深谷，機器人與太空船之間的通訊訊號有可能受到地上物的阻擋而無法通訊，因此如果機器人要與太空船通訊，它必須移動至可與太空船通訊的地點，此一地點我們稱它為可通訊地點。可通訊地點和太空船之間的連線必須沒有任何阻隔物，也就是說此點為機器人視線可及太空船之地點〔line-of-sight〕。機器人收集資料的方式如下：它將資料收集告一段落後，再尋找一個可通訊點進行與太空船的通訊。此問題的目的是為機器人設計一個程式，讓機器人在地表蒐集資料告一段落後，它可以找到一個最佳的可通訊地點與太空船通訊，此處所謂的最佳通訊地點乃是它所移動的距離必須是最短的，因機器人本身儲存的電力有限，必須儘快找出最接近的一點可以與太空傳通訊，因此每個機器人擁有數個 CPU，以便於最短的時間內利用存在記憶體中的地圖將最佳可通訊地點確定。以下我們針對本題所用的資料結構做一個簡單的描述。

地圖以 $n\times n$ 的矩陣 M 來表示，探勘地區座標$(i,\ j)$之地表最高點的數據存於矩陣第 i 列第 j 行 $M_{i,j}$。若 $M_{i,j}=(x_{ij},y_{ij},a_{ij})$，則表示探勘地區座標$(i,\ j)$的最高點位於$(x_{ij},y_{ij})$且其高度為 a_{ij}，a_{ij} 可以是一個負數或正數，$i\le x_{ij}<i+1$且 $j\le y_{ij}<j+1$。又已知探勘地區的地表高度變化緩和，因此定義相鄰兩

點的高度變化為線性的關係。總括來說，此探勘任務如下所述：太空船降落於地點 A，其座標為 (x_A, y_A)，並釋放出多個機器人，某機器人在蒐集資料告一段落後，位於地點 B，其座標為 (x_B, y_B)，機器人將以 (x_B, y_B) 為起點移動至最佳可通訊地點 C，其座標為 (x_C, y_C)，以便與太空船通訊。假設以 (x_A, y_A) 與 (x_C, y_C) 兩點之間拉成直線稱為 L_{AC}，則直線 L_{AC} 不應與地面物有任何交點，也就是直線 L_{AC} 經過的地點其地表高度皆應比該直線在空間的高度為低。本題之主要演算過程分兩部分，第一部分是以太空船之座標 (x_A, y_A) 為基礎，先行確認探勘地區之所有地點 R 是否符合可通訊地點之條件，並將所有符合條件之地點標示下來。第二部分則是以機器人目前所在位置的座標 (x_B, y_B) 為起點，以最短路徑演算法於所標示之可通訊地點中找出最佳的點。假設在一開始時，所有電腦模組的記憶體內皆存有一份矩陣 M，以下分兩部份說明我們的演算法。

第一部分：計算可通訊點的演算過程可從矩陣資料的切割著手以達成計算過程的平行化，我們先將矩陣資料切割成若干個條狀區域，再一一的將各區域分別指定給不同的 W 個電腦模組 $p_1, p_2, \ldots, p_i, \ldots, p_w$ 處理。矩陣資料的切割方式採用列切割，也就是將 $n \times n$ 的矩陣資料以列為基礎切個成 n 個條狀區域，每個區域有 n 筆資料。在不喪失一般性的原則下，我們假設 n 是 W 的整數倍，因此每部電腦將處理 n/W 個區域。實際上，我們的演算法亦適用於 n 不是 W 的整數倍。由於演算過程中，距離太空船 (x_A, y_A) 越遠的地點，其 line-of-sight 計算量將大於距離近的地點，原因將在步驟二中說明，所以我們發現每個條狀區域所隱含的運算量並不相同，離太空船越遠的區域，所需的運算量越大，因此影響此部分平行化效能的關鍵是條狀區域指定給叢集中各個電腦模組的方式。在此我們討論兩種指定方式：區塊指定(block allocation)與循環指定(cyclic allocation)，這兩種方式在平行處理技術中經常被用來處理負荷均衡(load balance)的問題。在區塊指定的方式中，每部電腦負責處理連續的區域，也就是將列 $((i-1)n/W)+1$ 至列 (in/W) 指定給電腦模組 p_i。在循環指定的方式中，每部電腦所處理的區域是不連續的，也就是將列 $i, W+i, 2W+i, \ldots, n-W+i$ 指定給電腦模組 p_i。因此若採用區塊指定的方式將會造成電腦模組運算量不均的狀況，採用循環指定則可以使得電腦模組的運算量較為均勻。

(1) 步驟一：求出太空船頂端(假設天線位於頂端)與所有其它地表最高點連成的三度空間直線方程式，以圖 4-a 為例，地表位置(4,5)之最高點 R 與太空船頂端 A 所形成的三度空間直線方程式 L_{AR}。此部分的運算很容易平行化，若有 W 部電腦與 $N(=n \times n)$ 個地表位置，則每部電腦模組負責 N/W 個直線方程式之求取。求取一個直線方程式所費的時間是固定的，總共有 N 條直線，所以 W 部電腦在此步驟的總運算量為 $O(N)$。

(2) 步驟二：此步驟是用來標示地表上所有的可通訊點，在此步驟中，探勘區域的所有地表位置將被檢查是否滿足 line-of-sight 的條件，每一部電腦模組負責以循環指定的 n/W 個區域內的所有地點。我們以圖 4-a 來說明某一地點 R 是否滿足 line-of-sight 條件的檢查方式，同樣的方式亦用於其它地點 line-of-sight 條件的確認。首先，我們必須求出三度空間直線 L_{AR} 在平面 $z=0$ 上投影所形成的二度空間直線經過的所有地點座標，也就是(1,1)(2,2)(2,3)(3,3)(3,4)(4,4)(4,5)，只要前述任一投影地點之地表高度超越三度空間直線 L_{AR}，則 R 不滿足 line-of-sight 條件。但要注意的是，先前

假設相鄰兩點的地表高度變化為線性，因此不宜直接取 $a_{11}a_{22}a_{23}a_{33}a_{34}a_{44}a_{45}$ 的值作比較。原因可由圖 4-b 看出來，圖中地點 Q 之最高點比三度空間直線 L_{AR} 低，但由 PQ 兩點構成且垂直於 z=0 的平面 S_{PQ} 卻與 L_{AR} 相交，因此只用地點 Q 之最高點來比較是不正確的。所以必須以投影所形成的二度空間直線所經過的地點之高度，分別與其周邊地點之高度來求得地形變化之規則，此規則有助於判斷 S_{PQ} 是否與 L_{AR} 相交。若相交則無法滿足 line-of-sight 條件，若地點 (1,1)(2,2)(2,3)(3,3)(3,4)(4,4)(4,5)皆滿足 line-of-sight 的條件，則地點 R 滿足 line-of-sight 的條件，並將其標示為可通訊地點。此部分的運算很容易平行化，若有 W 部電腦模組參與運算和 N 個地表位置，則每部電腦模組負責 N/W 個地表位置之 line-of-sight 的判斷。此部份會因 R 距離太空 A 越遠而需要檢查的投影點越多，需要檢查的投影點越多，代表運算量越大，這也就是我們為何資料分配時採用循環指定的原因。檢查一條直線是否滿足 line-of-sight 條件，其所費時間的上限與該條直線的長度成正比，最長的直線為對角線，因此檢查一條直線花費 $O(\sqrt{N})$，共有 N 條線要檢查，所以 W 部電腦在此步驟的總運算量為 $O(N\sqrt{N})$。

第二部分：搜尋最短路徑的演算過程不易平行化，因為運算之間都有前後關係的相依性，只能針對其中部分的運算採用平行化處理。影響此部分平行化效能的關鍵是彼此信息傳遞的部分，當電腦模組的數量增加，電腦模組間的通信量也會隨著增加，若通信量增加所造成的負擔超過平行運算所帶來的效益時，所耗費時間將可能隨著電腦模組的個數增加，因此依照矩陣的大小來選擇適當的模組數是必須的。

(1) 步驟一:求出地表上各點與其鄰點的地表距離，所謂地表距離就是機器人由某一地點走至另一地點的實際距離。我們以距離矩陣表示各點間的地表距離，若距離矩陣之第 u 列第 v 行之值為 w，則表示從地點 u 至地點 v 的距離為 w。假設某一點 u 的座標為(i,j)，我們必須求出 u 至點 $(i\text{-}1,j\text{-}1)(i\text{-}1,j)(i\text{-}1,j\text{+}1)(i,j\text{-}1)(i,j\text{+}1)(i\text{+}1,j\text{-}1)(i\text{+}1,j)(i\text{+}1,j\text{+}1)$的地表距離，並將此一數據存於距離矩陣。此部分的運算很容易平行化，若有 W 部電腦與 N 個地表位置，則每部電腦負責 N/W 個地表位置至鄰點的地表距離。求取一個地表位置至鄰點的地表距離，所費的時間是固定的，所以 W 部電腦在此步驟的總運算量為 $O(N)$。

(2) 步驟二:機器人找尋至最佳通訊地點的演算法是以 Dijkstra 演算法[18]為基礎並稍做修飾，Dijkstra 演算法一般用於求取某一起始城市至所有其它目的城市的最短路徑，在此我們可將目前機器人所在地當做起始城市而所有的座標點為目的城市，而最佳可通訊的地點是從所有標示為可通訊地點中選出離機器人所在地點最近者。實際上，我們可以不用將所有可通訊地點之距離求出，只要是最早被確定距離的可通訊地點，即是最佳通訊點。此修飾過之演算法與 Dijkstra 演算法一樣採用 Greedy 的策略來找尋最短路徑，它不斷的以目前已求得最短路徑的某些城市中挑選一個城市當跳板，以求取其它未知距離之城市的路徑或更新某些城市目前之路徑以符合最短距離之要求，由於此步驟每次都從以一個已知最短路徑的城市中藉以求得其它城市的最短的路徑，因此需要執行比較的動作，所以在比較各個路徑長短的過程中有平行的空間，其餘的則為循序的

步驟。我們以圖 5 作概略的說明，假設機器人目前位於地圖座標格(1,1)，以步驟一所求得的距離矩陣如圖 5-e 所示。在一開始，我們計算機器人目前位置至右方座標格、下方座標格及右下方座標格之距離，如圖 5-a 所示，計算結果如下：至右座標格(1,2)之距離為 1，至下座標格(2,1)之距離為 3，至右下座標格(2,2)之距離為 8。接著由目前獲得最短距離之座標格(1,2)開始對其周圍之座標格計算距離，也就是以(1,2)為起點計算由(1,2)至(1,3)(2,1)(2,2)及(2,3)的累加距離，所獲得的累加距離分別為 4、9、7 和 5，如圖 5-b 所示。座標格(2,1)及(2,2)在第一次計算和第二次計算時分別得到不同的值 3、8 和 9、7。在這些座標格選取較小的值，也就是設定座標格(2,1)為 3 及(2,2)為 7，如圖 5-c 所示。並從 (1,3)(2,1)(2,2)及(2,3)挑選最小值的座標格重覆上述步驟直到遭遇到可通訊地點，最先遭遇到的可通訊地點就選為最佳可通訊地點。若(2,1)為可通訊點，則其為最佳可通訊點，如圖 5-d 所示。因為 Dijkstra 演算法採用 Greedy 的技術，若繼續找其它點，其距離會比 3 更長。Dijkstra 演算法之時間複雜度較高，若每一地點之鄰點個數不超越某常數 C，Dijkstra 演算法之時間複雜度為 $O(CN+N^2)$，此題符合每一地點之鄰點個數不超越某常數 C，所以 W 部電腦在此步驟的總運算量為 $O(CN+N^2)$，此式可化簡為 $O(N^2)$。

我們以不同的地表面積來運算並求取數據，該地表高低的資訊存於 $n\times n$ 的矩陣，該矩陣在一開始時即儲存於每部電腦，其中 n 由 12 遞增至 240，將太空船設定在地圖之左上角，機器人設定在地圖之右下角。採用循環指定的策略，所得到之數據結果如圖 6 所示，所得到之數據結果顯示，只要資料量夠大，當電腦的個數從一個增加到三個時，執行時間呈現下降的趨勢，當啟動四部電腦執行此平行化程式時，執行時間反而開始上昇，主要原因是第二部分的演算過程，可被平行化的程式只佔了一部分而已，而且在目前獲得最短距離的數個座標格中選取最小值時，電腦模組之間必須做信息交換，以便互相比較目前存於各個電腦模組中的最小值，此信息交換的次數會隨電腦模組個數增加而增加，每次信息交換只傳送少量的資料，因此時間複雜度為 $O(W)$。從數據資料中可推得，四部以上的電腦彼此傳遞信息的耗費時間，已經大到使整體效能開始下降。我們亦將不同資料量之下的加速比呈現於圖 7。

此問題平行化之複雜度來自分割後之各區域的運算量分佈不易獲知，就是採用循環指定的方式將各區域分配給電腦，在檢查 line-of-sight 的條件時，極有可能只需檢查少數幾個投影點就可得知該地點不符合 line-of-sight 的條件，不必繼續檢查。改善以上所述造成各部電腦運算量不均的狀況，也許可採用動態負荷均衡(dynamic load balance)的策略[20]，當某部電腦閒置時再分配額外的資料給它。另外，我們也可以從修改演算法的某些步驟著手，利如 Dijkstra 演算法不易平行化，若我們能將該演算法輸入的資料量減小，則時間會縮短，可能的做法是在第二部份的第一步驟和第二步驟之間多加入一個處理步驟，此處理步驟之目的是將原先用來代表該探勘地區所有點之相鄰點間的距離矩陣轉成較小的矩陣，而此較小的矩陣是用來描述符合 line-of-sight 條件之各點間的距離。這樣的策略雖然增加了一個步驟，但此步驟易於平行化。經過這步驟處理後，Dijkstra 演算法可以花費較少的時間，也許整體的時間會減少。

伍、 結論與未來研究方向

1. 系統效能方面，雖然我們架構的規模小，使用四部電腦模組組合成小型的叢集系統，然利用既有經驗，組成大型的叢集系統不成問題。
2. 叢集系統的彼此通信順暢與否，會影響到整體的運算效能，因此將叢集系統的網路環境與其它區域網路，甚至與 Internet 的連線，最好有做隔離的處理。本次實作規模較小，受到的影響並不明顯，但將來可考慮另外架設一部有路由作用的伺服器做隔離。
3. 利用 NFS 及 NIS 可以整合 Linux 系統群的檔案共享及使用者帳號共享，在架設更多部電腦的叢集系統時，這是相當需要的。
4. 平行程式設計對於大專程度的學生確實有難度，在設計的過程中我們遇到許多的難題，希望透過我們電腦叢集系統的建置，使用於教學與專題研究，使學生能更熟悉平行程式設計的技術。
5. 我們對於電腦叢集系統與平行處理的技術已獲得了相當多的知識，未來希望透過不同領域專長人士的加入，進一步發展適用於叢集系統的平行演算法及開發大型應用軟體。

謝誌

本研究承蒙國科會學生專題計劃(NSC89-2815-C-197-017R-E)及國立宜蘭技術學院研究計劃經費補助暫行辦法之經費(NIIT90-006)支持，王煌城博士提供寶貴的意見，特此致謝。

參考文獻

1. H. J. Siegel, et al. (1992), “Report of the Purdue Workshop on Grand Challenge in Computer Architecture for the Support of High Performance Computing”, J. Parallel and Distributed Computing, Vol.16, No.3, pp.199-211.
2. J. C. Venter, et al. (2001), “The Human Genome Sequence”, Science, Vol.291, No.5507, pp.1304-1340.
3. IEEE Symposium Record – Hot Chips IV (1992).
4. http://www.ibm.com
5. http://www.intel.com
6. http://www.paracel.com
7. R. Hughes, “Quantum Computation (2001),” IEEE Computing in Science and Engineering”, Vol.3, No.2, pp.26.
8. W. S. Lin, et al. (2001), “Adaptive Parallel Rendering on Multiprocessors and Workstation Clusters”, IEEE Tran. on Parallel and Distributed Systems, Vol.12, No.3, pp.241-258.

9. A. Kalinov and A. Lastovetsky (2001), “Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers”, J. Parallel and Distributed Computing, Vol.61, No. 4, pp.520-535.
10. X. Du and X. Zhang (1999), “The Impact of Memory Hierarchy on Cluster Computing”, Proc. 13th Int’l Parallel Processing Symposium, pp.12-16.
11. G. R. Luecke, B. Raffin, and J. J. Coyle (1999), “Comparing the Communication Performance and Scalability of a Linux and a NT Cluster of PCs, a Cray Origin 2000, an IBM SP and a Cray T3E-600”, Proc. 1st IEEE Int’l Workshop on Cluster Computing, pp.26-35.
12. 張智豪、李丕榮、吳真貞 (2001), “於同質/異質的平行計算環境求解 Euler 方程式”, 中華民國資訊學會通訊, 第四卷, 第二期, 第 31-46 頁。
13. J. J. Dongarra and D. W. Walker (2001, May/June), “The Quest for Petascale Computing”, IEEE Computing in Science and Engineering, pp.32-39.
14. 楊朝棟, 張宏守 (2000), “在 Linux 上建構與應用叢集式平行電腦系統”, Linuxer, 第十二期, 第 88-102 頁。
15. http://www.netlib.org
16. I. Foster, and C. Kesselman, eds. (1999), The Grid: Blue Print for a Future Computing Infrastructure, Morgan Kaufmann.
17. http://www-unix.mcs.anl.gov/mpi/mpich
18. T. Leighton (1985), “Tight Bound on the Complexity of Parallel Sorting”, IEEE Tran. on Computers, Vol.34, No.4, pp.344-354.
19. M. A. Weiss (1999), Data Structures & Algorithm Analysis in Java, Addison-Wesley.
20. B. Wilkinson and M. Allen (1999), Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers, Prentice Hall.