分散式多柵程式的設計及其在平行虛擬機器上之執行

The Design of a Distributed Multigrid Algorithm and Its Execution on a PVM System

王煌城

宜蘭農工專科學校電子工程科

hcwang@ccmail.niiat.edu.tw

摘 要

本文探討多柵演算法在平行虛擬機器上執行的相關問題，包括分散式程式的設計與行爲分析、程序間資料交換的方式、程式執行過程的觀察與記錄、效益的評估等，並討論分散處理中的重要問題如資料分割與配置、程序間之負載平衡與同步，所設計之演算法在一由個人電腦與工作站構成之分散系統中執行的結果也在本篇中加以分析。

關鍵詞：多柵演算法、平行虛擬機器、單一程式多重資料、異質多重處理、區域分割法

Abstract

This paper investigates issues related to the execution of multigrid method on a parallel virtual machine (PVM). The effort involves the design of a distributed multigrid program and analysis of its behavior. Interprocess communications and data exchanges are examined. Data partitioning and mapping, load balancing, and synchronization are discussed in this context. SPMD paradigm of distributed processing is adopted. The distributed multigrid program is tested on a system consisting of heterogeneous workstation and personal computers. Empirical data verify the correctness of the program. Performance results and execution profile are presented with analysis.

Keywords: Multigrid, parallel virtual machine (PVM), SPMD, heterogeneous multiprocessing, domain decomposition

一、緒論

平行處理 (Parallel processing)與分散式處理 (Distributed processing)的技術在近幾年有長足的進步[1]，相關研究方向主要有二：其一是同質處理 (Homogeneous processing)，亦即利用多個功能相同的處理器，以解決一複雜的問題；另一爲異質處理 (Heterogeneous processing)，此爲同質處理之延伸，容許不同特性之機器彼此合作以解決一問題。同質處理吸引不少廠商從事開發，而 Intel 爲美國能源部所建立一含數千個 PentiumPro 處理器的平行處理系統，是其中最著名的例子之一。而在異質處理方面，亦有蓬勃的發展，例如 Condor [2]即爲利用工作站群組進行分散式處理，以取得高產出 (High throughput)的計畫，有 Wisconsin 大學、NCSA、台灣中研院等機構參與。

分散式處理牽涉多項課題，包括如何建立執行環境、如何設計演算法、如何選擇適當的執行模式、如何編寫程式碼、如何將欲執行的工作切割予不同的程序、如何將程序分配予不同主機等，皆屬複雜的問題，其中以合適程式碼的開發更爲當務之急。

本研究探討多柵 (Multigrid)演算法[3]在 PVM (平行虛擬機器)[4]環境下的執行，內容包含：PVM 環境的建立、PVM 程式的設計與編譯、偵錯、執行、效能分析等項。由於分散式處理程式設計較循序程式複雜，程式行爲亦不易掌握，故須充分運用適切的工具始能簡化程式偵錯工作，並能觀察程式的動態行爲。

分散式多柵程式的發展甚受重視，德國司徒加大學 (University of Stuttgart)的 UG 計畫爲一綜合性、跨科系的計畫，其演算法可以解不同類型的二維與三維空間問題。除了演算法本身功能甚爲強大外，該系統之軟體亦考慮到在各種不同的 UNIX 平台上作業環境的差異。至於分散式處理的支援則與慕尼黑大學(University of Muenchen)等機構合作，組成 SEMPA 計畫(Software Engineering Method for Parallel Applicationsin Scientific Computing)[5]，其目標之一爲進行 Algebraic multigrid 方法的分散式處理。

康乃爾大學 (Cornell University)之研究群由 Craig Douglas 主導，所開發完成之軟體爲 MADPACK，目前已推出 MADPACK 5，其中 MADPACK 2 [6]已有支援 MPI (Message Passing Interface)[7][8]標準之分散式處理能力，MADPACK 5 則仍在進行平行化設計階段。此軟體特點在於使用者可自行設定所欲解問題之特性 (二維或三維、柵之點數等參數)，亦可設定多柵演算法之屬性 (Relaxation、restriction、prolongation)。

美國 NIST(National Institute of Science and Technology)也支援一相關研究，該計畫名爲 PHAML (The Parallel Adaptive MultiLevel Project) [9]，其目的爲產生原有的循序多柵程式 MGGHAT

* 國科會研究計畫編號 NSC 84-2213-274-001。

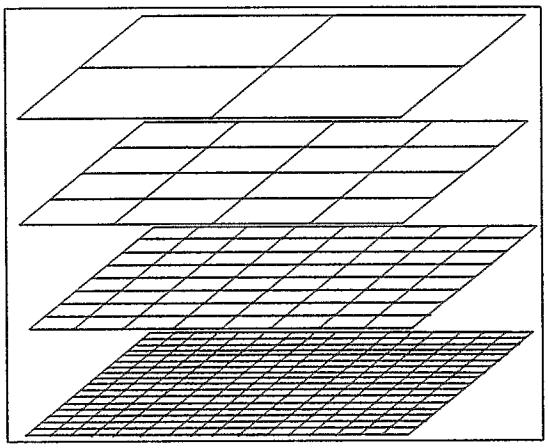
之平行處理版，以在分散式記憶多處理器系統如IBM SP2上執行。

UG、MADPACK、以及其他多柵演算法之資料可自MGNet獲得[10]。

二、分散式多柵演算法的設計

多柵演算法發展已有多年歷史，其基本原理乃利用一系列疏密不等的柵(如圖一所示)，以消除誤差中不同的頻率成分。這種方法與單柵演算法比較，收斂速度甚爲快速，故廣受科技界重視。

圖一：多柵演算法以一系列疏密(間距)不等的柵加快收斂速度



多柵演算法中的主要計算包括：

- 每一柵上以傳統方式，如 Successive Overrelaxation (SOR)、Gauss-Seidel iteration等求得近似解。
- 從疏柵到密柵的延伸 (Prolongation)或內插(Interpolation)計算。
- 從密柵到疏柵的限制 (Restriction)或投射(Projection)計算。

求近似解以及延伸、限制等方法選擇常視問題性質而定，也是許多研究的重點。本研究初步以二維 Poisson 方程式：

$$\nabla u = f \quad \text{in } \Omega = (0,1)\times(0,1)$$

配合 Dirichlet 邊界條件做爲測試，正確解爲 $u = x \sin \pi x \sin \pi y$。

爲達到分散處理的目標，多柵演算法須配合區域分割法 (Domain decomposition)並用，每一分割的小區域內的問題即由一程序負責求解，求解過程中程序需交換資訊。爲了讓每一程序負載約略相等，每一區域內的點數在分割時亦使之大略相同。以下就程式中同步點的設定與資料交換特性等加以討論，詳細程式請參考[11]。

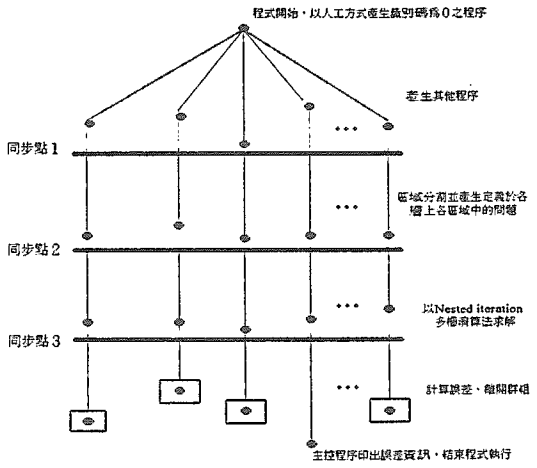
同步點的設定

同步點在分散式程式的執行上甚爲重要，如果在應該同步的地方未定義同步點，則容易產生錯誤。例如在圖二中，若未設定同步點一，則可能在某一程序未被產生前，已有其他程序欲與之交換資料，因而造成錯誤。

在一高度異質化的系統中，由於各主機效能差距頗大，執行指令速度亦有差別，很容易形成 race condition 或 deadlock；同步點的設置，可避免此種情況的發生。然而不必要的同步點則可能造成不必要的等待，因而降低處理的速度。

從圖二中可以看出，同步點將求解過程畫分爲個階段，主控程序最早產生、最晚結束。在各階段中，每一程序抵達同步點的時間可能不同，由圖中黑點所示，這相當於 PVM 系統中各主機效能上與負載分配上的差異，負載平衡的目的就是希望所有程序抵達同步點的時間能夠很相近。在最後階段，各程序將該區域誤差計算出來後，將值傳予主控程序，即離開群組；主控程序在收到區域誤差，完成加總後，結束全部程式之執行。

圖二：分散式多柵程式的同步點設定



PVM 中提供了定義同步點(Synchronization points)的指令，其格式如下：

pvm_barrier(group, ntask)

其中 group 爲群組名稱，ntask 則爲該群組中程序的數目，當所有屬於 group 的 ntask 程序均執行完此指令後，才能執行其後的指令。

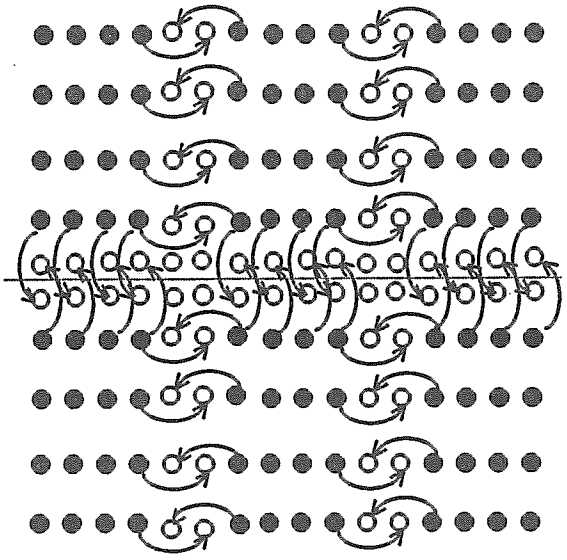
資料交換特性

在分散式多柵演算法中的資料交換可分爲三種類型：

A. 互換式通訊(Interchange)：發生於對應到相鄰區域的程序之間，如圖六所示。各程序在子區域周圍

有影子柵點(Shadow grid points)，在圖中以空心圓表示，以標明這些值是由臨近之程序(子區域)供應，與原屬該子區域之柵點(以實心圓表示)不同。此圖顯示垂直與水平方向的資料交換。這種類型的交換在多柵演算法執行過程中甚爲頻繁，各柵層上的求近似解 (Smoothing)，層間值的傳遞 (Projection 與 Prolongation)等運算均需要程序間互相交換邊界點的資訊，這種交換屬於點對點通訊。在程式設計上，要先以群組識別碼確定程序間的上、下、左、右之座標關係，始能決定那二者之間需要通訊。PVM 相關指令爲 pack 與 send 及 recv 與 unpack。

圖三：分散式多柵程式在各柵層上的資料交換



B. 散播式通訊 (Scattering)：此種方式意指一程序將同一資料散播予組群中之所有成員。在本程式中，主控程式 (群組識別碼爲 0 者)取得待解問題之資訊後，將該資訊廣播予所有的其他成員，即屬此型通訊方式，PVM 中提供 mcast(群播)與 bcast(廣播)二函式以執行此類資料交換。

C. 彙聚式通訊 (Gathering)：這種方式與散播式通訊相反，乃是群組中的成員將資料全部傳予一特定程序，屬於多對一的通訊。在分散式多柵程式中，當求解過程結束後，各程序分別計算所分配區域內的誤差值，再將該值送予主控程序彙整，以計算整體的誤差，即屬於此類通訊方式，PVM 的 reduce 函式支援此型運算，惟該指令須配合 freezegroup 指令使用方可。

群組識別碼與相對座標關係

在 PVM 系統中，屬於同一群組的程序均其群組識別碼(Group ID)，設群組中共有 N 個程序，則識別碼爲 0 至 N－1。群組識別碼爲 0 的程序是以人工方式產生，亦即在 PVM 或 XPVM 中以 Spawn 指令產生的程序，其他程序則由此程序產生；亦即群組識別碼爲 0 的程序是其他程序的父(Parent)程序，其他程序則爲子(Son)程序。一般而言，父程序負責群組共同功能，例如輸出、入指令的執行，並負責收集其他程序執行的結果。在 PVM 系統中，也可以經由群組識別碼取得對應的程序識別碼(Task ID)，以便用於資料送收等敘述。

群組識別碼的另一主要功能乃用以決定程序間的相對位置。以多柵演算法而言，如將某一柵層分爲 4×4 = 16 個子區域，每一子區域分配予一程序(群組識別碼爲 0 至 15)，則群組識別碼與子區域間相對座標位置關係如圖四所示。如果子區域座標位置以(r,s)表示，群組識別碼以 G 表示，則有 r = G / 4，s = mod (G, 4)的關係。

圖四：群組識別碼與子區域列、行編號(括號內數字)

12 (3,1)	13 (3,2)	14 (3,3)	15 (3,4)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)

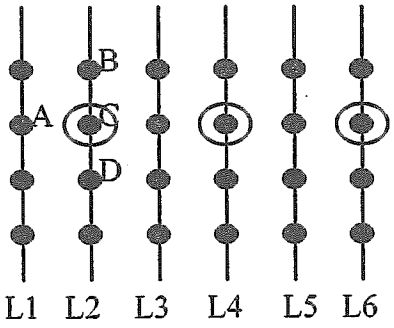
子區域相對座標 (r,s) 可用以決定那些子區域間須交換資料，爲重要參數。此種相對座標關係在 MPI 中稱爲 Virtual Topology ，並有特殊指令支持相關的資料交換；在 PVM 中則須以群組識別碼來決定。

區域分割法應考慮的因素

在分散式多柵演算法的設計中，區域分割是一重要的步驟，將每一柵層加以分割爲數個子區域，並將每一子區域分配給不同的程序，則數個程式可以同時處理定義於不同子區域上的計算，達到平行處理的目標。

執行區域分割須考量二項因素，一爲負載的平衡，一爲不同柵層之間的聯繫。就負載的平衡而言，每一子區域內的柵點數應力求相近，這是因爲計算的複雜度與柵點數成正比。以二維矩形柵而言，其做法爲將每一維度之柵點數除以在該維度之程序數；若無法整除，則將餘數平均分配予相同數目的程序，詳細的分割方法請參考 dcmp1d (One-dimensional decomposition)副程式[11]。如此所得到的區域分割方法可以確保各子區域中的柵點數約略相等，達到負載平衡的目標。

圖五：不同柵層區域分割時的考量



就不同柵層間的聯繫而言，多柵演算法在由一密柵轉移至疏柵時，須執行 projection；反之，由疏柵至密柵則要執行 prolongation 運算。Projection 一般採 Full weighted 方式，如圖五所示。圖中。代表密柵上的柵點，○代表疏柵上的柵點。如果在疏柵上做區域分割時，L1 與 L2 被分配給程序 1，而在密柵上，L2 與 L3 被分配予程序 2，則當要執行 Full weighted prolongation 時，程序 1 上將缺乏足夠的資料以進行必要的計算。這是因爲在程序間的資料交換(亦即相鄰子區域間的資料交換)只限於邊界上的柵點(程序 1 將 L1 上柵點的值傳予程序 2，程序 2 將 L2 上柵點的值傳給程序 1)，致計算疏柵上 A 點之值所需密柵上的 L3 之 B、C、D 三點的值無法取得，影響演算法的正確性。此時即應對分割方式加以修正，使疏柵上 L2 改爲分配給程序 2，這可能會對負載平衡稍有影響，但其效果甚爲輕微。

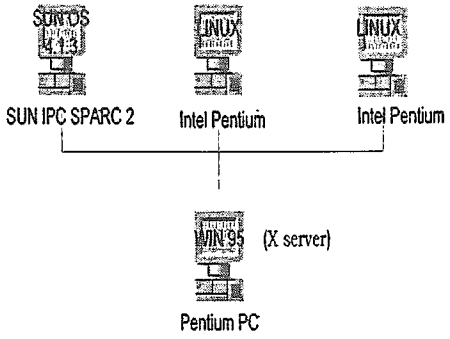
區域分割對演算法收斂速度的影響

由於區域分割將某些原爲相鄰的柵點分配給不同的程序，故可能對演算法的收斂性質產生影響。舉例而言，如果在各柵層上採用 Gauss-Seidel relaxation 或其他立即取代法(Immediate replacement methods)以求取近似解，則在各子區域邊界上的柵點在計算新值時所用的鄰點值乃是原先的舊值而非已更新過的值。這是因爲相鄰子區域間的資料只發生在 relaxation 進行前與完成後的緣故，因此分散式演算法的收斂速度會與循序演算法有差異。

三、測試環境

本研究以由 Sun SPARC 工作站與個人電腦(PC)構成之平行虛擬機器做爲測試平台。系統各主機之間以 Ethernet 互相串接。系統之控制則爲另一 PC，該 PC 上執行 X Window 之伺服軟體，PVM 主機則做爲 X 視窗之 client 端，整體系統架構如圖六所示。

圖六：PVM 執行環境



表一：PVM 系統各主機主要特性

機器名稱	CCMAIL	Wang	Wang1
CPU 類型	SUN IPC SPARC 2	Pentium 133	Pentium 133
主記憶容量	48MB	16MB	16MB
硬碟容量	6GB	1GB	1GB
作業系統	SUN OS 4.1.3	LINUX 2.0	LINUX 2.0
編譯程式	g77,f77,gcc	g77,f77,gcc	g77,f77,gcc

在此 PVM 系統中，Sun Sparc 主機採用 Big endian 的位元組序(Byte order)，而 Intel Pentium 則採取 Little endian[12][13]，二者構成一異質處理系統，故主機間交換的資料須經編解碼，以確保正確的送收；程序間的訊息傳遞乃使用 TCP 協定[14]。

四、執行結果與討論

收斂速度的比較

圖十顯示循序與分散式多柵程式執行時，收斂速度的比較。多柵演算法特性如下：

基本迴圈結構：巢狀迴圈(Nested iteration) V cycle。

各柵層上之近似解：Gauss-Seidel 執行二次。

疏柵到密柵資料轉移：Bilinear interpolation。

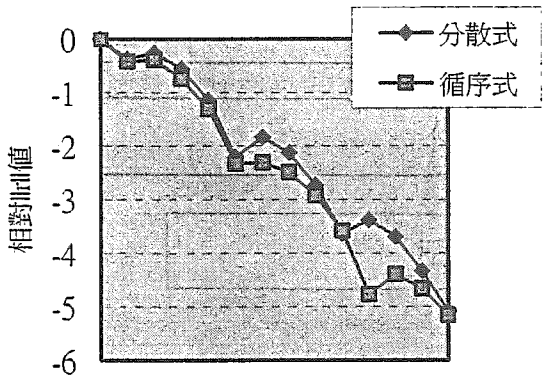
密柵到疏柵資料轉移：Full-weighted projection。

最密層柵點數：17 x 17。

分割之子區域數(分散式處理)：4。

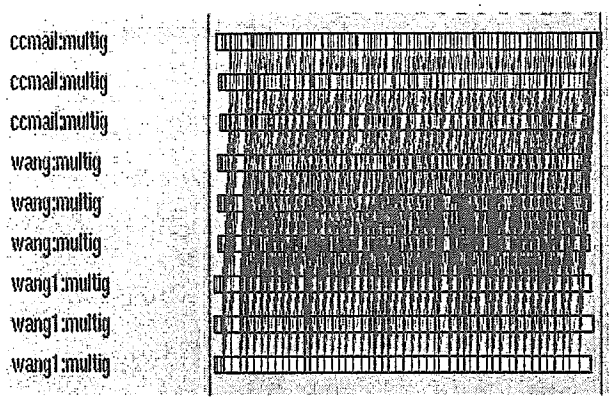
圖十爲一 semilog 圖形，垂直方向代表餘數向量值(Residual norm)相對誤差(以對數表示)隨著多柵演算法迴圈而變化的情形。從圖中可看出二者收斂速度的差別，如所預期的，分散式演算法收斂速度稍遜於循序程式。

圖七：循序與分散式多柵演算法收斂速度之比較



圖八顯示在執行過程中，PVM 系統中各主機進行計算與交換資料的情形。本次執行中一共有九個程序平均分佈在三部主機上，彼此間依據演算法需求交換資料。從圖中可以看出資料交換的頻率非常高且均勻，事實上依此 space-time view 圖形來看，大部份的時間皆花費在通訊上，計算所費時間所佔比例較低，這是由於本例所解 Poisson 方程係用點數 33*17 的柵(最密層)，故計算時間甚短所致。從圖八中可以看出 ccmail 由於速度比其他兩部爲慢，而主控程序是在 ccmail 上，因此整體執行時間亦受影響。

圖八：PVM 系統中各主機執行計算與資料交換情形



表二列舉在 PVM 系統上執行一次完整多柵演算法周期所需時間(以秒爲單位)，第一欄的問題大小係以最密柵上的點數來表示。此處之執行時間係以主控程序(群組識別碼爲 0 者)的執行時間爲準，這是因爲主控程序最早產生、最後結束，相當於圖六中的最長路徑(Critical path)。

表二：在不同大小的柵上執行一次完整多柵演算法所需時間

Problem Size	Levels	Number of Processes (domains)	User Time	System Time
33×17	2	9	0.47	1.00
33×33	3	9	0.65	1.57
33×65	3	9	0.78	1.75
65×65	4	9	1.14	3.00

表中時間分爲使用者時間與系統時間，前者乃指實際之計算時間，後者則爲資料傳送與其他等待時間。此表數據顯示，真正用於計算的時間約佔全部時間的三成；而且執行時間並未隨著問題大小呈線性成長，這有利於解決較大型的問題，惟須注意主機記憶體容量等限制。

五、結 論

本研究主要成果在設計出一分散式處理的多柵程式，並測試、分析該程式在一由工作站與 PC 構成的 PVM 系統上執行的結果。此程式利用區域分割法將各柵層的處理工作分配予不同的程序以達到平行處理的目標。本文中討論了程式設計上應注意的因素，以及程式在執行時的行爲。

此程式稍加修改即可用以解決更複雜的問題，也可在功能更強大的 PVM 系統上執行，從而得到效能上的改善。這種情況隨著高速工作站與 PC 的普及，加上網路速度的提昇，很容易即可實現。另一效能的可能改進來自演算法的改善，從本實驗的結果以及演算法的特性分析，可以發現在程式執行的過程中有很頻繁的資料交換，若能將小段訊息結合爲長訊息，即可降低通訊時間，提高執行效能，這也提供了演算法一個研究的方向。

另一方面，實驗所需之執行環境的建立頗爲費時。由於 PVM 系統要求參與之主機上均要執行 PVM Daemon，故每一主機上均需有完整的作業環境，包括PVM軟體的建置以及各種語言編譯程式，且需注意不同主機上的程式版本須維持一致，否則即會出現意想不到的錯誤，這些往往造成系統發展上的困擾，分散式物件技術或可提供一解決方案。

誌 謝

本研究進行期間，承蒙 XPVM 開發者 Jim Kohl 與 Madpack 設計者 Craig Douglas 的協助，特此申謝。

參考文獻

[1] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, New York, 1993.

[2] A Worldwide Flock of Condors: Load Sharing among workstation clusters, Journal on Future Generations of Computer Systems, Vol. 12. 1996

[3] C. C. Douglas, Multigrid Methods in Science and Engineering, *IEEE Computational Science and Engineering,* vol. 3, no. 4, pp. 55-68, 1996.

[4] A. Geist et al., PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing, The MIT Press, 1994

[5] P. Luksch,, U. Maier, S. Rathmayer, and F. Unger. Software Engineering Methods for Parallel Applications in Scientific Computing-Project SEMPA. *IEEE Concurrency*, July-September 1997.

[6] C.C. Douglas, MADPACK 2 Users' Guide, IBM Research Report RC 16169, Yorktown Heights, 1990.

[7] Message Passing Interface Forum, MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS—94-230, University of Tennessee, Knoxville, TN, April 1994. (Also appeared in the International Journal of Supercomputer Applications, vol. 8, no. 3/4, 1994).

[8] I. T. Foster, Designing and Building Parallel Programs, Addison-Wesley, Reading, MA, 1995.

[9] W.F. Mitchell, The Full Domain Partition Approach for Parallel Multigrid on Adaptive Grids, proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, to appear.

[10] C. C. Douglas and M. B. Douglas, "MGNet Bibliography," Yale University, New Haven, Conn., 1991-96. Available on MGNet at *http://casper.cs.yale.edu/mgnet.www/mgnet.html.*

[11] 王煌城，多柵與共軛梯度演算法之分散式處理研究，國科會研究計畫報告，1997。

[12] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1990.

[13] W. Stallings, Computer Organization and Architecture: Designing for Performance, Prentice-Hall, 1996.

[14] J. Postel, Transmission control protocol. RFC 793, Information Sciences Institute, September 1981.