階層式交叉立方體之三回合適應性診斷模擬

孫崧閔, 賴寶蓮
國立東華大學資訊工程系
Email:songmiin1989@gmail.com
baolein@mail.ndhu.edu.tw

摘 要

在分散式系統中，要如何即時、正確、有效的診斷出壞掉的處理器是一項重要的議題，因此本論文以階層式交叉立方體 (Hierarchical Crossed Cubes) 網路為模擬對象，並且根據過去提出的三回合診斷演算法在 MPICH2 平行計算平台上進行模擬。原本在階層式交叉立方體上故障的處理器只要不超過維度的個數，都能利用這套診斷演算法在第三回合內找出所有故障的處理器，本論文把故障處理器提高到兩倍的維度的個數，並利用 MPICH2 平行計算平台分別一一探討，在不同故障個數下用這診斷演算法所發生的情形。

一、簡介

系統級的診斷 ($system-level\ diagnosis$) 在提高系統可靠性和可用性，是一項重要的理論。PMC model 是由 Preparata, Metze and Chien [13] 提出的一個系統級錯誤診斷的圖形理論模型。在此模型中，系統中的每個處理器都會進行測試，通過其鄰居之間的通訊聯繫。當一個處理器測試另一個時，測試者宣告被測試處理器是好的或壞的，取決於測試的反應。如果測試者是好的，測試結果是準確的，但如果測試者是壞的，結果是不可靠的。

Nakajima [11] 首次提出適應性診斷 ($Adaptive\ Diagnosis$)，以已知的測試結果為基礎，可動態地確認之後的測試。這方法提出一重要觀點是盡可能地確認一個無故障的處理器，然後用它來診斷系統中其他的處理器。Hakimi 和 Nakijima [3] 提出了一種適應性診斷演算法，在完全圖上，假設錯誤的數量最多 t 個，測試次數最多 $|V|+2t-2$ 次，便可確定所有故障的處理器。

適應性診斷是根據蒐集到的症狀來判斷處理器是否為好 ($fault-free$)、壞 ($fault$)。適應性系統級診斷演算法進行的測試回合，在週期時間內，每個點會被分配到已執行的測試中，經過數回合的測試才能確定該點的狀態。通常使用的兩個方式來評估適應性演算法為 1. 每一回合的測試數和 2. 總測試數和診斷延遲。

階層式圖形結構 [2, 6, 8, 10, 12, 16] 在多處理器系統 (multiprocessor system) 上，有相當多的研究；階層式圖形結構通常是由一個或多個圖形結構所組合而成，階層式交叉立方體是由超立方體與交叉立方體形成的網路結構 [9]，本論文以 $HCC_{k,n}$ 表示一個由 Q_k 和 CQ_n 所組成的階層式交叉立方體，$HCC_{k,n}$ 擁有良好的性質，例如它是正規圖 ($regular$) 且自由度 ($degree$) 為 $n+k$，有 2^{2n+k} 個節點，容錯性高，相當適合應用在大型的多處理器系統上 [3]。

MPICH2 [14, 17] 是建構於電腦叢集之上的平行計算平台，基於訊息傳遞介面 (Message Passing Interface) 的架構而成，將訊息傳遞介面簡稱為 MPI，MPI 是第一個標準化的平行化應用界面，它的函式庫提供許多傳輸與溝通用的函式，開發人員能直接挑選合適的函式進行平行程式的研發，不必再花費額外的力氣自行撰寫，目前 MPI 介面已由 MPI 協會於 1998 年更新至 2.0 版本，MPICH2 即以 MPI2.0 為基礎所開發，至今一直保持版本的更新，具有免費與高可攜度的優點，也同時支援 Linux 環境和 Windows 環境，因此我們選擇使用 MPICH2 做開發模擬的研究。

接下來的論文中，分成五個章節，首先第二章會再更詳細介紹階層式交叉立方體，第三章為介紹診斷演算法，第四章為說明 MPICH2 架構下的診斷演算法設計，第五章是實驗模擬結果，最後第六章是本論文的結論。

二、階層式交叉立方體

階層式交叉立方體，是結合超立方體與交叉立方體而成的一個網路結構，既保有了兩種圖形結構的特性之外，自由度為 $n+k$，節點個數有 2^{2n+k}，邊數為 $(n+k)*2^{2n+k-1}$ 此圖為正規圖，相當適合應用在大型的多處理器系統上。階層式連結網路結構於多重處理系統 ($multiprocessorsystems$) 中，有相當多的研究 [5, 9, 10, 15, 16]。

本論文接受國科會編號:NSC 102-2115-M-259-007 研究計畫經費補助

定義 **1** *[9]* 當 $n > 0$ 且 $k > 0$ 的階層式交叉立方體，記作 $HCC_{k,n}$，是一個有 2^{2n+k} 個節點的圖形，其 $V = \{b_{2n+k-1}b_{2n+k-2}\cdots b_1b_0 | b_i \in \{0,1\}, 0 \le i \le 2n-1+k\}$。一個節點 u 的編號，$u = u_{2n+k-1}u_{2n+k-2}\cdots u_1u_0$ 可分成三個層級，分別為 X_u 層級、Y_u 層級和 Z_u 層級*:*

$$X_u = u_{2n+k-1}u_{2n+k-2}\cdots u_{2n}$$

$$Y_u = u_{2n-1}u_{2n-2}\cdots u_n$$

$$Z_u = u_{n-1}u_{n-2}\cdots u_0$$

此外，邊集合 E，可分成兩個子集合，分別為 E_{int} 和 E_{ext}，E_{int} 集合代表內部邊，E_{ext} 集合代表外部邊，定義如下列等式**:**

(1) $E_{int} = \{(u,v) | X_u = X_v, Y_u = Y_v,$ 且 $(Z_u, Z_v) \in E(CQ_n)\}$、

(2) $E_{ext} = \{(u,v) | Y_u = Z_v, Z_u = Y_v,$ 且 $(X_u, X_v) \in E(Q_k)\}$。

根據定義 1，$HCC_{k,n}$ 網路結構包含了三個層級：

- 下層 (Z level)：有 2^{n+k} 個交叉立方體我們稱 sub-CQ_n 或 $CQ_n^{(i,j)}$，其 i、j 定義與中層 (Y level) 相同。
- 中層 (Y level)：可分成 2^k 個群組，記作 $S_{HCC_{k,n}}(i), 0 \le i \le 2^k - 1$，每個 $S_{HCC_{k,n}}(i)$ 包含了 2^n 個 sub-CQ_n，每個 $S_{HCC_{k,n}}(i)$ 群組中的 sub-CQ_n 也記作 $CQ_n^{(i,j)}$，$0 \le j \le 2^n - 1$ 。將一個二進制字串 b 轉成十進制 (Decimal) 的表示，記作 $dec(b)$，其 $CQ_n^{(i,j)}$ 的 i、j，則為 $CQ_n^{(i,j)}$ 中的節點之 $u_xu_yu_z$，$i = dec(u_x)$ 和 $j = dec(u_y)$。例如：任一節點 $u = 11000001$，為 $HCC_{2,3}$ 圖中某節點，$dec(11) = 3$ 和 $dec(000) = 0$，清楚知道此點 u 為 $CQ_3^{(3,0)}$ 群組中的某一節點。

  此外，在 $S_{HCC_{k,n}}(i)$ 群組中 sub-CQ_n，彼此之間不會有邊相連；換句話說，$S_{HCC_{k,n}}(i)$ 群組中 sub-CQ_n，會跟 $S_{HCC_{k,n}}(j)$ 群組中 sub-CQ_n 有一條邊互相連接，此邊稱為外部邊。
- 上層 (X level)：類似一個超正立體 Q_k，此外 $S_{HCC_{k,n}}(i)$ 群組，代表超正方體結構中的一個節點 i。

如圖 1為 $HCC_{1,1}$ 的圖形結構，圖 2為 $HCC_{1,2}$ 與 $HCC_{2,1}$ 圖形結構。

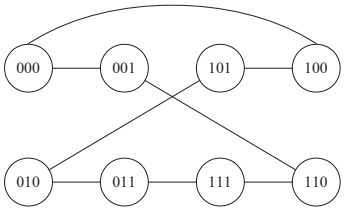

圖 1: $HCC_{1,1}$

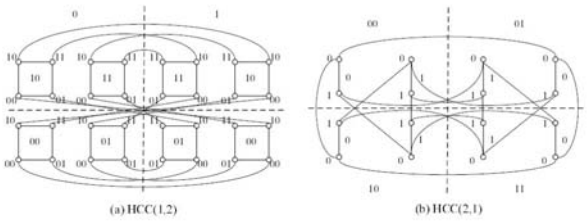

圖 2: (a) 為 $HCC_{1,2}$ 網路結構，(b) 為 $HCC_{2,1}$ 網路結構

三、診斷演算法

如果一個系統包含所有的處理器，當故障的處理器不超過 t 時，均可辨識，我們稱此系統為 $t-diagnosable$。

我們用一個無向圖 $G(V,E)$ 來模擬多處理器系統的拓樸，其中在一集合 V 中的點代表處理器，在邊集合 E 中的邊代表處理器之間的聯繫；因此，點可以透過邊測試任何相鄰的點。令點 u 和點 v 在圖形上是一對相鄰的點，點 u 對點 v 測試，記作 $r(u,v)$，其中點 u 為測試者，點 v 為被測者。若點 u 測試點 v 的結果是好的 (fault–free)，則 $r(u,v) = 0$；若點 u 測試點 v 的結果是壞的 (faulty)，則 $r(u,v) = 1$。注意，測試的結果不一定是可靠的，如果測試者 u 是壞的，則測試結果可能不正確。如表 1經過測試得到的結果我們稱為「症狀」(*syndrome*)。

測試者	受測者	測試結果
好	好	0
好	壞	1
壞	好	0或1
壞	壞	0或1

表 1: PMC 模型的測試規則

本論文所採用之三回合診斷演算法 [7]，已在 2013 發表。只要故障的處理器不超過 $k+n$ 個，都能在三回合內找出所有故障的處理器。本論文把故障處理器提高到最多 $2(k+n)$ 個，並探討壞 $(k+n+1) \sim 2(k+n)$ 個處理器時，此診斷演算

法上的診斷能力。此演算法，會先在 $HCC_{1,n}$ 中建立交替迴圈 (*interleaved cycles*)，交替迴圈是一種特殊的迴圈，$HCC_{1,n}$ 圖形結構下層為交叉立方體 (CQ_n)；交替迴圈只經過每個 CQ_n 中兩個相鄰節點所連結的內部邊一次，和經過兩個相鄰的 CQ_n 所連接的外部邊一次，由內部邊與外部邊交錯而成的迴圈。如圖 3是一個 $HCC_{1,3}$ 的其中一條交替迴圈建構例子。接著在交替迴圈上進行兩回合基本測試並收集症狀，再根據症狀決定進階測試的安排，以期在三回合內找出所有錯誤。

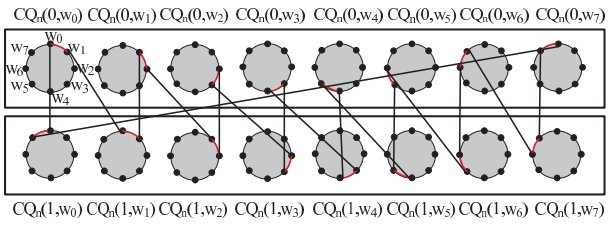

圖 3: $HCC_{1,3}$ 的交替迴圈

在進行第三回合測試前，我們會先用到 $Cycle-partition$ 和 $Path-colour$ 這兩個演算法進行分割與辨識，兩個演算法表示如下。

Algorithm 1 $Cycle-partition$ [1]

Step1:Choose a 0-arrow a_0 followed by a 1-arrow.
Step2:Let a be the arrow following a_0.
If a is 0-arrow, then set a_0 a and proceed step 2;
otherwise, go to step 3.
Step3:Mark with an X the arrow following 1-arrow a.
If it was not already marked, set a_0 the X-marked arrow and go to step 2;
otherwise, the algorithm is terminated.

Algorithm 2 $Path-colour$ [1]

1:Let m denote the maximum number of fault(s), s denote the number of the paths obtained by $Cycle-partition$, and let $q=m-s+1$. We give a colour to each vertex in each path as follow .
2:If a path has more than $q+1$ vertices and has at least one 0-arrow, we colour the head of the path with black and colour $q-1$ vertices from the tail with grey, and all vertices that are not coloured yet are coloured white.
3:If a path has at most $q+1$ vertices, then we colour all vertices in the path with grey.

四、MPICH2 架構下的診斷演算法設計

當開始執行一個 MPI 程式時，會根據使用者的設定創建相應數量的程序，每個程序都代表一個運算工作，這些程序會被分配到各台計算機，同時運算、同時結束；在開始進行運算時，需先取得或計算負責的工作範圍，工作切割方式依問題特性而定，常見且容易劃分的是資料切割，若欲做運算分割則必須謹慎確認各運算步驟之間的前後關係，工作範圍確立後，即可開始進行核心運算作業，在運算過程中程序之間若需要交換資料，MPI 有提供 1 對 1 及 1 對多 (廣播) 的傳輸函式，若每個程序都需讀取相同的檔案，則該檔案須放置公用資料夾中，才能被共同存取，待所有程序的運算都結束之後，挑選一個程序統一收集其他程序的運算結果，再做最後的結果輸出，接著根據設定的 k 和 n 值，MPI 環境的 *processes* 數量為 2^{k-1}，完成前置動作即可開始執行 MPI 程式。

在這裡說明一下演算法裡的 f_i 表示單一 *process* 裡有問題的交替迴圈數量，f_c 則表示全部 *process* 裡的 f_i 加總起來的值，整體演算法簡述如 Algorithm 3, 其中進階測試共分為 3 個 *cases*, 若 *process* 內的所有點均是好的, 此 *process* 被稱為好的 *process*, 否則稱為壞的。另外,$P0$ 代表編號 0 的 *process*。

- Case 1. $f_c \geq (k+n-1)$。
- Case 2. $f_c \leq (k+n-2)$ 和 $f_i \leq (n-1)$。
- Case 3. $f_c \leq (k+n-2)$ 和 $f_i \geq n$。

Algorithm 3 整個 MPI 平行演算法
如下輸入: HCC 的 k 和 n 的值
輸出: 所有診斷結果

1: 根據輸入的 k 和 n 值亂數產生 $2(k+n)$ 以內個錯點;
2: 利用交替迴圈演算法產生 2^{n-1} 個症狀;
3: 利用產生的症狀計算 f_i 的值;
4: 計算 f_c 的值;
5: **if** $f_i \neq 0$ **then**
6: 對每一個有問題的 *interleaved cycle* 執行 $Cycle-partion$ 演算法和 $Path-colour$ 演算法;
7: **end if**
8: **if** $f_c \geq (k+n-1)$ **then**//case1
9: 利用交替迴圈自行判斷即可;
10: **else**
11: 把每個 *process* 上 f_i 的值傳到 $P0$;
12: **if** *process is* $P0$ **then**
13: 判斷要執行 *case*2 或 *case*3;
14: **end if**
15: $P0$ 利用廣播告訴每個 *process* 第三回合診斷要執行 *case*2 或 *case*3;
16: **if** *process* 要執行 *case*2 診斷方法 **then**
17: 利用 CQ_n 配對並進行測試診斷;
18: **else**//case3
19: **if** *process is* $P0$ **then**
20: 配對壞的 *process* 編號與好的 *process* 編號;
21: **end if**
22: 根據配對傳送壞的 *process* 裡的可疑點給好的 *process*;
23: 好的 *process* 收到壞的 *process* 裡的可疑點;
24: 好的 *process* 回傳測試診斷結果給壞的 *process*;
25: 壞的 *process* 收到診斷結果;
26: **end if**
27: **end if**
28: 各個 *process* 印出診斷結果;
29: 結束

步驟 1，程式執行時會根據輸入的 k 和 n 值亂數產生 $2(k+n)$ 以內個錯點。
步驟 2，接著程式會為每個交替迴圈的來產生一個基本測試結果 (共 2^{n-1}) 稱為症狀。
步驟 3，每個 *process* 會計算自己的 f_i 值。
步驟 4，會利用 MPI 裡的 $MPI\_Allreduce$ 指令把每個 *process* 的 f_i 值全部加總起來，並且把加總的值存放在每個 *process* 的 f_c 上。
步驟 5~7，如果各自 *process* 的 f_i 值只要有出現大於等於 1 就會執行 $Cycle-partion$ 演算法和 $Path-colour$ 演算法。
步驟 8~9，為第三回合診斷方法的 case1。
步驟 11，每個 *process* 會把自己的 f_i 值傳到 $P0$ 裡面。
步驟 12~14，$P0$ 會根據上一步驟傳的 f_i 值有無大於等於 n，來判斷第三回合應該統一執行 case2 或 case3。
步驟 15，$P0$ 它會利用 MPI 裡的 MPI_Bcast 指令來告知全部 *process* 第三回合診斷應該統一執行 *case*2 方法或 *case*3 方法。
步驟 16~17，為第三回合 *case*2 的診斷方法。
步驟 18~26，為第三回合 *case*3 的診斷方法。
步驟 28 會印出各個 *process* 裡面好點或壞點的診斷結果。

五、實驗結果

本次實驗的 MPICH2 的模擬環境，主機為 i7-3770 處理器，記憶體 8GB，硬碟 500GB，作業系統是 Windows7 64 位元，MPICH2 版本為 1.4.1p1。

以下模擬結果中，t 代表的是好點，f 代表的是壞點，eu 代表的是無法被診斷出來的可疑點，tf 表示原本是好點，但診斷後判斷為壞點，ft 表示原本是壞點，但診斷後判斷為好點。以下圖中, 縱軸代表 100 次實驗後的平均 tf、ft, 與 eu, 橫軸代表錯點數量。

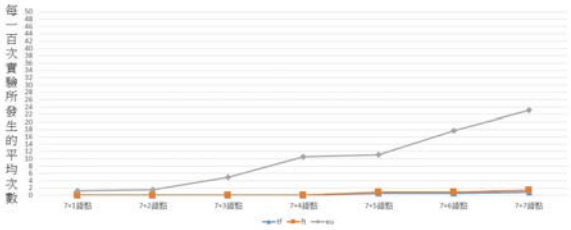

圖 4: $HCC_{1,6}$ 的實驗數據線性圖

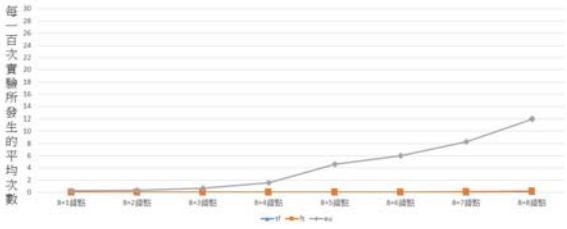

圖 5: $HCC_{2,6}$ 的實驗數據線性圖

綜合以上結果可以發現當 k 和 n 值越來越大的時候，eu 明顯的下降，我們的討論結果是，由於 k 變大所以使得錯點分布可以更廣，只要錯點不集中在某一塊區域，可疑點就能被附近的好

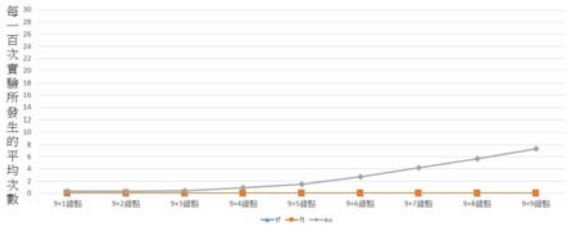

圖 6: $HCC_{1,6}$ 的實驗數據線性圖

點偵測出來，所以 eu 會明顯下降。而 ft 容易發生的情況，就是當 n 很小時候，或是當錯點數量太多時候，原因是根據 $case1$ 我們可以知道每個 $HCC_{1,n}$ 可以切割成 2^{n-1} 條交替迴圈，每條交替迴圈頂多會錯一個點或是錯兩個點，但是由於我們測超過原本可容忍的錯點量，所以變成每條交替迴圈會有可能錯超過兩個點。tf 這情況也會發生，但發生機率不會比 ft 高。以下圖中, 縱軸代表時間 (sec), 橫軸代表總點數。

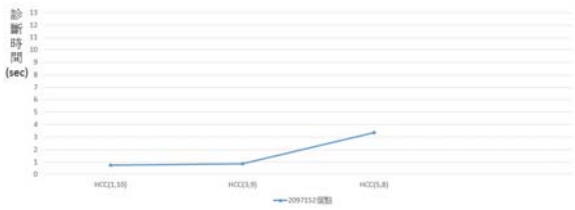

圖 7: 診斷 2^{21} 個點的時間線性圖

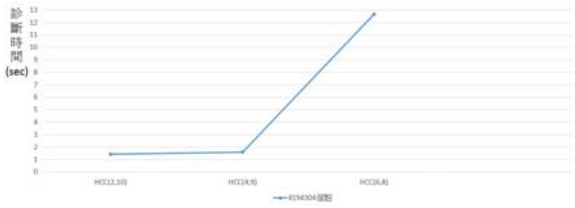

圖 8: 診斷 2^{22} 個點的時間線性圖

由上面線性圖可以清楚知道，當我們在總點數不變情況下，任意改變 HCC 的 k 和 n 的值，發現當 k 越大，也就是 $process$ 數量越來越多的時候，我們診斷花費的時間比較長，最後發現原因是 $MPI\_Allreduce$ 的這個指令，它讓每個 $process$ 都參與了數字加總，所以當 $process$ 數量越多的時候，在這上面所花費的時間就更大了，但即使診斷總點數 2^{23} 個點，大約將近 800 萬個點數量，也還是能在半分鐘內跑完，因此本演算法具有即時性。

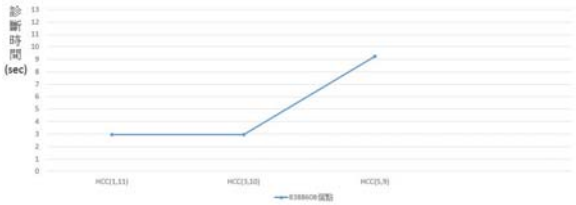

圖 9: 診斷 2^{23} 個點的時間線性圖

六、結論

在 $case1$ 中，整個測試結果上發現，當 HCC 的 n 太小，或是錯點數量太多情況下，會令我們的 tf、ft 和 eu 這些數值增加，而當我們的 k 和 n 越來越大的時候，就算錯兩倍的點數量，這三個數值還是會不斷的降低，甚至趨近於 0，而 $case2$ 中，則是可以利用好的 CQ 鄰居來測所有可疑點，和 $case3$ 中，一定可以找到至少一個好的外部 $process$(表示整個 $HCC_{1,n}$ 裡的點都是好點) 去測有可疑點的 $process$(表示整個 $HCC_{1,n}$ 裡至少會有一個以上的可疑點)，所以 $case2$ 和 $case3$ 的 tf、ft 和 eu 的數值都是 0。最後在效能方面上，本篇論文在產生 2^{23}(約 800 萬) 個點去執行這套演算法還是能在一分鐘內跑完，大致整體速度而言還算可行。

參考文獻

[1] S. Fujita and T. Araki, "Three-round adaptive diagnosis in binary n-cubes," *Lecture Notes in Computer Science*, vol. 3341, pp. 442-451, 2004.

[2] S. Horiguchi and S. Fukuda, "A hierarchical redundant cube connected cycles for wsi yield enhancement," In Proceedings. *IEEE International Conference Wafer Scale Integration*, pp. 163-171, 1995.

[3] S. L. Hakimi and K. Nakajima, "On adaptive system diagnosis," *IEEE Transaction on Computers*, vol. C-33(3) , 234-240, 1984.

[4] S. Horiguchi and T. Ooki, "Hierarchical 3d-torus interconnection network," In Proceedings of ISPAN'2000. *IEEE CS Press*, pp. 50-56, Dec. 2000.

[5] V. K. Jain and S. Horiguchi, "Vlsi considerations for tesh: A new hierarchical interconnection network for 3-d integration," *IEEE Transactions on very large scale integration(VLSI) system*, vol. 6, pp. 346-353, 1998.

[6] J. M. Kumar and L. M. Patnaik, "Extended hypercube: A hierarchical interconection network of hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 45-47, 1992.

[7] P. L. Lai, M. Y. Chiu, and C. H. Tsai, "Three Round Adaptive Diagnosis in Hierarchical Multiprocessor Systems," *IEEE Transactions on Reliability*, vol. 62, no. 3, pp. 608-617, Sep. 2013.

[8] T. Y. Lee, P. A. Hsiung, and S. J. Chen, "Tcn: scalable hierarchical hypercubes," In Proceedings. *Parallel and Distributed Systems. Ninth International Conference*, pp. 11-16, 2002.

[9] P. L. Lai, H. C. Hsu, C. H. Tsai, and I. A. Stewart. "A class of hierarchical graphs as topologies for interconnection networks, " *Theoretical computer science*, vol. 411 (31-33), pp. 2,912-2,924, 2010.

[10] Q. M. Malluhi and M. A. Bayoumi, "The hierarchical hypercube: A new interconnection topology for massively parallel systems," *IEEE Transactions on Parallel and Distributed System*, vol. 5, pp. 17-30, 1994.

[11] K. Nakajima, "A new approach to system diagnosis," *Proc 19th Annual Allerton Conf Commun, Contr and Compu*, pp. 697-706, Sept. 1981.

[12] K. Padmanabhan, "Hierarchical communication in cube-connected multiprocessors," In Proceedings. *Distributed Computing Systems, 10th International Conference*, pp. 270-277, 1990.

[13] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosis systems," *IEEE Trans. Electronic Computers*, vol. 16, no. 12, pp. 848-854, Dec. 1967.

[14] M. J. Quinn, *PARALLEL PROGRAMMING in C with MPI and OpenMP*, 1st ed.Singapore:McGraw-Hill,2004.

[15] R. Y. Wu, G. H. Chen, Y. L. Kuo, and G. J.Chang, "Node-disjoint paths in hierarchical Hypercube networks," *Information Sciences*,vol. 177, pp. 4,200-4,207, 2007.

[16] S. K. Yun and K. H. Park, "Hierarchical hypercube networks for massively parallel computers," *Journal of Parallel and Distributed Computing*, vol. 37, pp.194-199, 1996.

[17] 鄭守成 (民 91)。C 語言 MPI 平行計算程式設計。取自 http://mx.nthu.edu.tw/jyhsu/Documents/mpic_2002.doc