中華民國八十六年全國計算機會議

多階層網路上環接任意節點之方法
LOOPING RANDOMLY SELECTED NODES ON MULTISTAGE INTERCONNECTION NETWORKS

李湘豪　林 偉
國立中興大學資訊科學研究所
台中市國光路 250

摘 要

在許多的平行應用程式中，multicast 是一種很重要的運算，例如一些平行搜尋和圖形演算法，或是單一程式多重資料的情形都會運用到這種運算。然而由於硬體的限制，往往是藉由多次的 unicast 來達到 multicast 的效果。倘若有一群節點彼此要做 multicast 的動作，也就是 group multicast，這將會需要許多的傳遞次數，使得傳遞資訊的時間大量增加。爲了加速這種運算，通常利用管線的技巧來完成資料的傳送，環狀結構則正好提供了實際的解決方法。Wormhole routing 是現今大多數的多重處理器系統所採用的資訊傳遞機制，只要所傳遞的路徑沒有被佔用，就可以不斷地將收到的 flit 送往下一個節點，如此使得傳遞延遲不會受到傳遞距離的嚴重影響。可是，一旦所需的路徑被佔用，這些傳遞的資料往往會被 block 在之前的路徑當中，直到所需的路徑被釋放。因此，傳遞的路徑彼此發生衝突時，會增加許多額外的傳遞延遲，所以要求每條傳遞的路徑彼此互斥是相當重要的。

在本篇論文中，我們提出了一個系統化的方法，來環接多階層交連網路上被隨機選擇的節點。這些被隨機選擇的節點就是要做 group multicast 的節點，我們將要安排一個環狀次序，使這些節點能順利地以管線的方式來完成 group multicast 的運算。這個方法的基本概念是由最小的環連接起，接著把小的環合併成較大的環，並且在每個步驟中都要確保完成的環中的每條路徑皆是互斥的。

Abstract

In this paper we present a systematic method of configuring group-multicast rings with randomly selected nodes on multistage interconnection networks. The method configures rings only with disjoint links so that the selected nodes can perform multicast communication without message collisions. The proposed method presents two unique features. First, it guarantees that the physical distance between adjacent nodes is limited to *n* links on an n-cube. Secondly, it uses only a single dimension order to route messages across the links towards their respective destinations.

Keywords: Computer networks, Data communication, Interconnection networks

1. 簡介

在多重處理器的系統當中，處理器節點常常需要藉著資訊傳遞(message-passing)的方式做彼此間的溝通。而其通訊型態(communication pattern)大致可分成 one-to-one、broadcast and multicast[1-3]。其中，multicast 是將相同的資訊傳送給任意個目的節點[4]。這種通訊方式在平行計算越來越被需要。支援 multicast 的系統可以提供較好的效能、較強的功能以及簡化程式設計，因其允許更高階的資料搬移運算[5]。

group multicast 更在許許多多的平行應用程式中被運用到，它是個非常重要的通訊操作。處理器節點常常藉由這種運算達到彼此的資料交換。然而，只要將處理器節點適當地設定，就可以利用管線方式的技巧來加速完成這種運算動作。例如有 n 個處理器節點需要做 group multicast 運算，經過我們將處理器節點做妥善地安排之後，利用管線方式來傳送資訊，則只要在 n-1 次的資訊傳送，各個處理器節點都將可以獲得所需的資訊。

對於這些需要做 group multicast 運算的處理器節點，想要加速其資訊傳遞的時間，使用管線方式是最有效的方法。也就是要讓在這一個群組中的處理器節點，能夠同時遞送出自身的資料給下一個節點，當每個處理器節點收到資料時，除了保留此資料外，再將這份資料傳給下一個節點。如此依著相同方向傳遞資料，可使得傳送資料的時間重疊，而達到管線的效果。由此可見，將這些要做 group multicast 運算的處理器節點安排成環狀的連接方式是達成管線方式傳遞資料的方法。

以管線方式來加速處理是非常便利的方法，可是，也將面臨到資源競爭問題。在交連網路中的資源則是通道(channel)。一旦要傳送資料的處理器節點間的路徑所需的通道互相衝突(conflict)時，則會使得管線方式的傳送停止下來，直到這個衝突解決爲止。解決這類資源衝突的方法，則會因每種網路架構的流量控制機制而有所不同[2]。當衝突發生時，不但不會加速資料的傳送，還會增加過度的傳遞延遲，甚至還可能引發死結(deadlock)[4,6]的情況，使得要放棄此次的資料傳送。可見，要避免這些處理器節點間的路徑所佔用的通道發生衝突是非常必要的。至於本篇論文所選擇的交連網路，是一個一般化、整合性和擴充性皆不錯的多階層交連網路 (multistage interconnection network)。並且以最常見的 Omega 網路架構爲代表[7]。

2. 背景知識及基本定義

2.1 多階層網路多重處理器模型

多重處理器系統有許多的處理器節點，這些節

點時常需要彼此通訊，而這些要傳遞的資料，便是經由多階層網路來送達目的節點。多階層網路的每個階層有許多個交換元件，藉由交換元件的設定，可以使得任何一個輸入節點到達任何一個輸出節點。

Omega 網路

一個 N x N 的 Omega 網路是由 L= $\log_2 N$ 個相同的階層(stage)所組成。每個階層則是 N/2 個交換元件以 perfect shuffle 的方式連接而成。如 Fig. 1 所示。

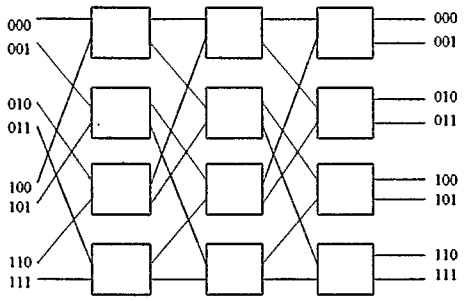

Fig. 1. 一個 8x8 Omega 網路

2.2 運算定義

定義 1

低位元運算是一種位元運算，用來求出兩個位址中，低位元位址相同的部分。這個↓數學位元運算定義了兩個位址的最長相同字尾，並且將其他的位元補上＊。

舉例來說，給定 X = 000101，Y = 001101，則 X↓Y = ＊＊＊101。

定義 2

高位元運算是一種位元運算，用來求出兩個位址中，高位元位址相同的部分。這個↑數學位元運算定義了兩個位址的最長相同字首，並且將其他的位元補上＊。

舉例來說，給定 X = 000101，Y = 001101，則 X↑Y = 00＊＊＊＊。

3. 基本定理

前面我們定義了一些位元運算。這部分就是藉由這些運算及 Omega 網路繞線的特性，來發展一些基本定理。這些定理包括如何判斷兩條路徑斥、如何建造出更多的互斥路徑等等。另外，本節中還介紹 buddy 子網路，以及如何將兩個子網路中的路徑連接起來。這些都是此篇論文所提出的演算法的基礎。

3.1 互斥定理

首先要提出一個定理，對於給定的兩條路徑，判斷它們是否互斥。Fig. 2 中可以看到在 Omega 網路中，起始節點 X 到目的節點 Y 的過程。由於在每一個階層中，都可能會和其他的輸入輸出配對發生衝突，所以，我們要找到一個方法，能夠較方便卻又不忽略期間因爲址的變化而產生的衝突。在 Omega 網路的繞線演算法中，一定要經過 L = $\log_2 N$ 個階層，每個階層又分成兩個小步驟，先做 shuffle 的動作，再經由交換元件選擇要走的路線。因此，若兩條路徑會在某個階層發生衝突，不是在做在 shuffle 的動作時發生，就是在經過交換元件時發生。在前面已提過，交換元件在 Omega 網路中只存在兩種狀態，分別是 straight 和 interchange。

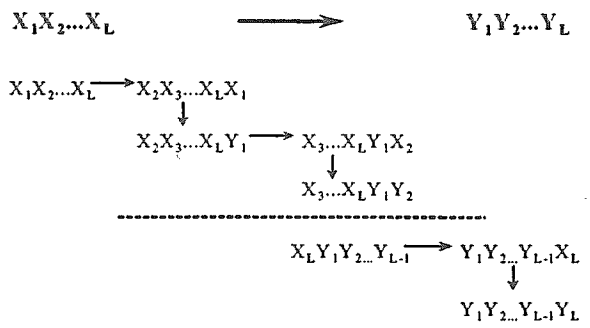

Fig. 2. 節點 X 到節點 Y 的位址變化

很明顯的，若這兩條路徑到達一個 2x2 交換元件的不同輸入端，不論此交換元件是何種狀態，都不會在交換元件這個步驟中發生衝突。

在交換元件這個步驟中要發生衝突，除非是兩條路徑到達同一個交換元件的同一個輸入端。若是這種情形，表示這兩條路徑在之前的 shuffle 步驟已經有衝突了。因爲要到達交換元件的同一個輸入端，必定是走相同的連接而來。在 Fig. 1 的 Omega 網路架構圖可以看到，這些階層間的連接(links)就是爲了做 shuffle 的動作，且其連接方式是固定的。因此，若兩條路徑佔用了相同的連接，表示其在此階層要做 shuffle 的動作之前，已經是到達相同的位址了。由此可知，我們可以用較方便的方法，來檢查兩條路徑是否會有衝突發生。這個方法就是在每一個階層開始做 shuffle 之前，其實也就是在前一個階層完成之後，來檢查兩條路徑是否到達相同的位址，若是，則此兩將會發生衝突。

在之前的 Fig. 2 中，節點 X 到節點 Y 的位址變化，其中，橫向箭頭是代表做 shuffle 的動作的位址變化，而縱向箭頭是代表經過交換元件的位址變化。我們要檢查的方式就是在每一個階層完成後，來比對兩條路徑是否有到達相同的位址。因此，就是要比對經過交換元件的位址。以下就是我們提出的判斷兩條路徑是否會互斥的定理。

定理一

在 NxN 的 Omega 網路之中，L=$\log_2 N$。Path(X,Y)和 Path(W,Z)互斥，若且唯若，

$$Ast(X \downarrow W)+Ast(Y \uparrow Z) > L。$$

證明：

節點 X 到節點 Y 的位址變化過程如 Fig. 2 所示，前面我們提到，兩條路徑 Path(X,Y)和 Path(W,Z)要互斥，則其每一階層完成時的位址都不能相同。Path(X,Y)在到達第 i 個階層位址會變成 $x_i x_{i+1} \ldots x_L y_1$

$y_2 \ldots y_{i-1}$，而 Path(W,Z)的位址會變成 $w_i w_{i+1} \ldots w_L z_1 z_2 \ldots z_{i-1}$。因此，以下條件要成立：

$x_i x_{i+1} \ldots x_L y_1 y_2 \ldots y_{i-1} \neq w_i w_{i+1} \ldots w_L z_1 z_2 \ldots z_{i-1}$

其中 $1 \leq i \leq L$。

此即 $Ast(X \downarrow W)+Ast(Y \uparrow Z) > L$，得證。

我們來看個例子，Path(2,6) 和 Path(6,5)。

$(X \downarrow W) = (010 \downarrow 110) = * 1 0$

$(Y \uparrow Z) = (110 \uparrow 101) = 1 * *$

由於 $Ast(X \downarrow W)+Ast(Y \uparrow Z)=2+1=3$，並沒有大於 L，我們由定理一可知這兩條路徑會發生衝突。

以下我們要說明一些預備知識，有了這些觀念之後，將會有助於了解我們所要提出的演算法。我們所要提出的演算法是將任意選擇的處理器節點排列成一個環狀次序，使之同時在做資訊傳遞時，這些路徑都不會發生衝突。這個演算法基本上是由連接最小的環開始，再將小的環合併(merge)成大的環，直到被任意選擇的處理器節點都包含在這個環中便完成了。如此，就可以利用管線的方式在最少的傳遞次數下完成 group multicast 的工作。此演算法是由最小的環的連接開始，也就是我們不會從整個網路下手，而是將網路分成最小的子網路(sub-network)，先完成各個子網路環的連接，再和其他子網路的環合併，以變成較大一點的環。如此下去，最後便可完成整個環的連接。由於是由子網路著手，這將會牽涉到 buddy 子網路的觀念，以下就先介紹何謂 buddy 子網路。

3.2 Buddy 子網路

定義 3

兩個 k 位元子網路，C_1 和 C_2。假設對於每一個子網路 C_1 中每一個的節點 X，都會存在一個屬於子網路 C_2 中的節點 Y，使得 X 和 Y 只有在由低位元開始第 k+1 個位址位元不同，其他處的位址位元皆相同。

以下就在 Fig. 3 中列出 16x16 的 Omega 網路中的 buddy 子網路的情形，我們可以在圖中看到每個 buddy 子網路之間上下的階層(hierarchy)關係。

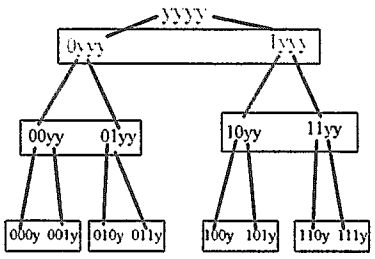

Fig. 3. 16x16 的 Omega 網路中的 buddy 子網路

在介紹了 buddy 子網路之後，我們再來看看 buddy 子網路的特性。buddy 子網路有一個重要的性質，就是各個子網路中的路徑是彼此互斥的。例如有兩個 buddy 子網路，C_1 和 C_2，則是不會和 C_2 中的路徑發生衝突的。但是要注意的是，這裡所說 C_i 中的路徑，是指起始節點和目的節點都屬於 C_i，i=1 or 2。這個特性是無論哪一種網路架構皆成立的，只要選擇適當的繞線演算法。

有了這個特性，我們便可以各自在不同的子網路中找尋互斥的路徑，由於 buddy 子網路中的路徑是互斥的，我們只要想辦法把這兩個 buddy 子網路的路徑再連接起來，就可以得到更大的互斥路徑集合了(disjoint path sets)。然而，重點是要如何選擇這兩個 buddy 子網路中的節點，來連接這兩個子網路。我們勢必要建立跨越子網路邊界的路徑，才能將兩個分離的子網路連接在一起，可是，隨意地建立路徑將會和已存在的路徑發生衝突。所以，我們要有特殊的方法來選擇要連接的節點，才能使造出來的路徑也和這兩個子網路中的路徑互斥。

假設有兩個 buddy 子網路，C_1 和 C_2。他們各自有自己的互斥路徑集合。我們要各自在 C_1 和 C_2 中找到一條路徑，假設在 C_1 和 C_2 中分別是 Path(A,B)和 Path(C,D)。這兩條路徑的起始節點，也就是 A 和 C，它們相同的低位元位址要是最長的，即 A 和 C 的位址有最長的相同字尾(postfix)。換句話說，這兩個 buddy 子網路中的其他路徑配對的節點，都沒有比 A 和 C 有更長的相同低位元位址。這兩條路徑被選出來之後，我們交換他們的目的節點，這個動作其實是建立 Path(A,D)和 Path(C,B)，並且取代原來的 Path(A,B)和 Path(C,D)。如此，便將 C_1 和 C_2 的路徑連接起來了。Fig. 4 可以看到建立路徑的情形。

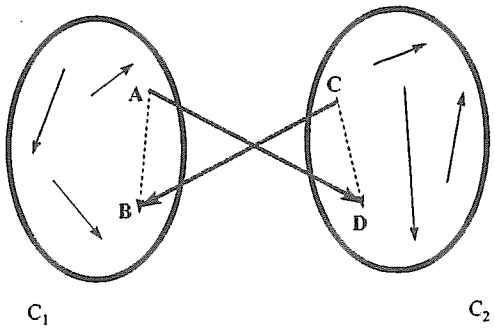

Fig. 4. 兩個 buddy 子網路的連接

接著，我們必須證明這種建立路徑的方法不會和原有的路徑發生衝突。但是，在這之前，我們要先提出其他有關的定理。定理一是告訴我們如何判斷兩條路徑是互斥或有衝突發生，以下的這些定理有助於我們建立互斥的路徑。

定理二

在 NxN 的 Omega 網路之中，$L=\log_2 N$。若 Path(A,B)和 Path(C,D)互斥，則 Path(A,D)和 Path(C,B)亦互斥。

證明：

由定理一可知，若 Path(A,B)和 Path(C,D)互斥，即

Ast(A↓C)+Ast(B↑D)>L。因爲↑運算有交換性，所以，(B↑D)=(D↑B)。因此，Ast(A↓C)+Ast(D↑B) = Ast(A↓C)+Ast(B↑D)>L。
即 Path(A,D)和 Path(C,B)互斥。得證。

讓我們來看一個前面的例子。在 8x8 Omega 網路中，Path(2,6)和 Path(4,5)互斥，因爲 Ast(010↓100)+Ast(110↑101)=2+2=4>3。現在將這兩條路徑的目的節點對調，則變成 Path(2,5)和 Path(4,6)。此時，Ast(010↓100)+Ast(101↑110) =2+2=4>3，可見，目的節點對調後，原本互斥的兩條路徑仍然是互斥的。

定理三

X,Y,Z 是 L 個位元的位址，若 Ast(X↓Y)≧Ast(X↓Z)，則 Ast(X↓Y)≧Ast(Y↓Z)。

證明：

令 $X=x_1 x_2 \dots x_L$，$Y=y_1 y_2 \dots y_L$，$Z=z_1 z_2 \dots z_L$，

$X↓Y=**\dots\dots*x_i\dots x_L$，

$X↓Z=**\dots*x_j\dots\dots x_L$。

因爲 Ast(X↓Y)≧Ast(X↓Z)，故 i≧j。可知 Y 和 Z 的低位元位址至少有 i 位元相同。因此可以假設，$Y↓Z=**\dots*y_k\dots y_L$，k≦i。

又由於 $X↓Y=**\dots\dots*x_i\dots x_L=**\dots\dots*y_i\dots y_L$。

所以，Ast(X↓Y)≧Ast(Y↓Z)。得證。

定理四

在 NxN 的 Omega 網路之中，$L=\log_2 N$。若 Path(A,B)和 Path(E,F)互斥，若存在一個節點 C，使得 Ast(C↓E)≧Ast(C↓A)，則 Path(C,B)和 Path(E,F)亦互斥。

證明：

由定理一，若 Path(A,B)和 Path(E,F)互斥，則 Ast(A↓E)+Ast(B↑F)>L。

由定理三，若 Ast(C↓E)≧Ast(C↓A)，則 Ast(C↓E)≧Ast(A↓E)。所以，

Ast(C↓E)+Ast(B↑F)≧Ast(A↓E)+Ast(B↑F)>L

即 Path(C,B)和 Path(E,F)互斥。得證。

3.3 子網路相連定理

有了這些定理之後，我們現在便可以回到先前提出連接兩個 buddy 子網路的方法，證明這樣建立路徑，並不會和已存在的路徑發生衝突。

定理五

假設在 NxN 的 Omega 網路之中，$L=\log_2 N$。令 C_1 和 C_2 是兩個 k 位元的 buddy 子網路，{Path1}是 C_1 中互斥路徑的集合，{Path2}是 C_2 中互斥路徑的集合。假設 Path(A,B) ∈{Path1}，Path(C,D) ∈{Path2}，且對於所有的路徑配對，Path(X,Y)∈{Path1}，Path(W,Z) ∈{Path2}，Ast(A↓C)≦Ast(X↓W)皆成立。則 Path(A,D)、Path(C,B)以及{Path1}和{Path2}中的路徑皆彼此互斥。

證明：

假設 Path(E,F) ∈{Path1}，E≠A，且 Path(G,H) ∈{Path2}，G≠C。

這個證明分成三個部分：

(1)先證 Path(A,D)與 Path(E,F)互斥

因爲 D 是 C_2 中的節點，而 F 是 C_1 中的節點，所以，這兩個節點必定在由低位元數來第 k+1 個位址位元會不同。可知(D↑F)這個運算結果是最低的 k+1 位元都是*，即 Ast(D↑F)=k+1。A 和 E 是 C_1 中的不同節點，可知它們在低位元開始第 k 個位元之前會有不同的位址位元，故 Ast(A↓E)>L-k。

Ast(A↓E) +Ast(D↑F)>(L-k)+(k+1)=L+1。

即 Path(A,D)與 Path(E,F)互斥。

(2)證明 Path(A,D)與 Path(G,H)互斥

因爲對於所有的路徑配對，Path(X,Y) ∈{Path1}，Path(W,Z) ∈{Path2}，Ast(X↓W)≧Ast(A↓C)皆成立。所以，Ast(A↓G)≧Ast(A↓C)。而 Path(C,D)和 Path(G,H)都屬於{Path2}，代表 Path(C,D)和 Path(G,H)是互斥的。由定理四，Path(C,D)和 Path(G,H)互斥，存在一個節點 A，

Ast(A↓G)≧Ast(A↓C)，

則 Path(A,D)與 Path(G,H)互斥。

(3)證明 Path(A,D)和 Path(C,B)互斥

Path(A,B)和 Path(C,D)是不同 buddy 子網路中的路徑，所以此兩條路徑彼此互斥。我們由定理二中得知，若 Path(A,B)和 Path(C,D)互斥，則 Path(A,D)和 Path(C,B)亦互斥。

同(1)和(2)，我們可以證明 Path(C,B)和{Path1}及{Path2}中的路徑互斥。綜合以上的證明，Path(A,D)、Path(C,B)以及{Path1}和{Path2}中的路徑皆彼此互斥。得證。

4. 演算法

4.1 演算法及實例

這個演算法的基本想法是由最小的子網路著手，先個別完成小的環，再和其 buddy 子網路的環合併，以成爲較大的環。如此不斷地用相同方式合併下去，直到包含所有被選擇的處理器節點爲止。這樣，就完成了將任意選取節點安排環狀次序的動作了。當然，在合併的每一個步驟中，都要保證這個環中的路徑是彼此互斥的，這也是這個演算法的基本原則。以下是演算法。在 NxN 的 Omega 網路，$L=\log_2 N$。

演算法

For k = 1 to L DO

  For 所有的 k 位元子網路{$x_1 x_2 \dots x_{L-k} yy\dots y$}，$x_1 x_2 \dots x_{L-k}$∈{0,1} DO

    IF 這個子網路中包含兩個被選取的分離節點 A 和 B

    THEN 建立 Path(A,B)和 Path(B,A)

    ELSE

    IF 這個子網路中包含一個環和一個單一被選取的節點 C

THEN 在這個環中找到 Path(A,B)，使得 Ast(A↓C)≦Ast(X↓C)，其中 X 是環上的節點；
建立 Path(A,C) 和 Path(C,B) 以取代 Path(A,B)
ELSE
IF 這個子網路中包含兩個環
THEN 在兩個環中找到路徑配對 Path(A,B)和 Path(C,D)；
使得對於這兩個環中所有的路徑配對 Path(X,Y)和 Path(W,Z)
Ast(A↓C)≦Ast(X↓W)皆成立；
建立 Path(A,D) 和 Path(C,B) 以取代 Path(A,B)和 Path(C,D)
FI FI FI
OD OD

這個演算法是由最小的子網路，也就是包含兩個節點的 1 位元子網路開始，依次檢查每個子網路，然後依照情形做不同的動作。此演算法基本上是由兩層的 For 迴圈所構成，在一個 NxN 的 Omega 網路中，$L=\log_2 N$，最小的子網路是 1 位元的，而最大的子網路，也就是整個網路，是 L 位元的。因此，外層的 For 迴圈是由 1 到 L。至於內層的迴圈的次數則會因外層迴圈 k 這個變數而有不同，內層迴圈所做的事是檢查每個 k 位元的子網路，並且完成適當的動作。因爲有 2^{L-k} 個不同的 k 位元子網路，這個演算法可以平行處理這 2^{L-k} 個步驟。

從演算法中可以看到，對於每一個要處理的子網路，會判斷 3 種情形而採取不同的動作。這也是演算法中 3 個 IF 敘述，如下：

(1) 如果子網路中恰好有兩個被選取的節點，則建立兩條路徑，將這兩個節點相連成長度爲 2 的環。
(2) 如果子網路由一個單一被選取的節點和一個環所組成，則將此節點合併到這個環之中。
(3) 如果子網路中包含兩個環，則將這兩個環合併成較大的環。

讓我們從 1 位元的子網路開始。在這個步驟時，1 位元的子網路只可能會存在 3 種情形，即這個子網路中沒有被選取的節點、有單一個被選取的節點、兩個節點都被選取。在我們的演算法中，前兩種情形都跳過不處理，而進入下一個步驟，也就是檢查 2 位元的子網路。只有第 3 種情形會採取演算法中(1)的動作，將兩個節點相連成長度爲 2 的環。因此，在完成第 1 個步驟，也就是對 1 位元子網路的檢查之後，這些 1 位元的子網路會存在 3 種情形，即子網路中沒有被選取的節點、有單一個被選取的節點、或是有一個長度爲 2 的環。

接著進入第 2 步驟，我們著手於 2 位元的子網路。一個 2 位元的子網路是由兩個 buddy 1 位元子網路所組成的。前面提到，1 位元的子網路存在 3 種情形，分別是沒有被選取的節點、有單一個被選取的節點、或是一個環。因此，兩個 buddy 1 位元子網路將會產生 6 種組合情形，即有 2 個被選取的節點、一個環和一個單一被選取節點、兩個環、一個環、一個單一個被選取的節點、或是都沒有節點被選取。換句話說，在 2 位元的子網路中，便會存在此 6 種狀況。

演算法中只對這 6 種情形中的前 3 種採取動作，而這(1)、(2)和(3)種動作的結果都是產生一個環；對於後面的 3 種情形則不做任何改變。如此，在這個步驟完成時，每一個 2 位元子網路將只有 3 種情形，即沒有被選取的節點、有單一個被選取的節點、或是一個環。這和第 1 個步驟完成後，每個 1 位元子網路所有的情形相同。由此可以證實我們只處理前 3 種狀況是可行的，因爲其他的狀況會在之後的步驟得到處理。如此，依相同的方式檢查每個 k 位元子網路，並採取適當的動作。直到 k =L 時，也就是第 L 個步驟之後，此演算法便完成。

以上我們可以確定，藉由這個有系統的方法，可以將任意被選取的節點連接成環狀。可是，重要的是，在這些過程中，是否完成的環中的路徑會彼此互斥呢?這方面的理論證明我們之後會證明。在這之前，讓我們回到{0,2,3,5,6}這個例子，來觀察這個環在每個步驟是如何形成的，並先以這個例子來驗證，是否依此方法完成的環中的路徑真的會彼此互斥。以下 Fig. 5 就是每個步驟完成後的連接情形。因爲是在 8x8 的 Omega 網路上，$L=\log_2 8=3$，所以只要 3 個步驟就可以完成環的連接。由圖中可以清楚第看到，第 1、2 個步驟分別是對每個 1 位元子網路以及每個 2 位元子網路做處理，直到第 3 個步驟，則是針對包含 8 個節點的整個網路。此步驟將兩個 2 位元子網路的環做合併便完成了。由以上的過程，可以看到{0,2,3,5,6}這 5 個節點的環狀次序將是{0,3,2,5,6}，也就是做 group multicast 通訊時，將同時在 0→3，3→2，2→5，5→6，6→0 這五條路徑上通訊。

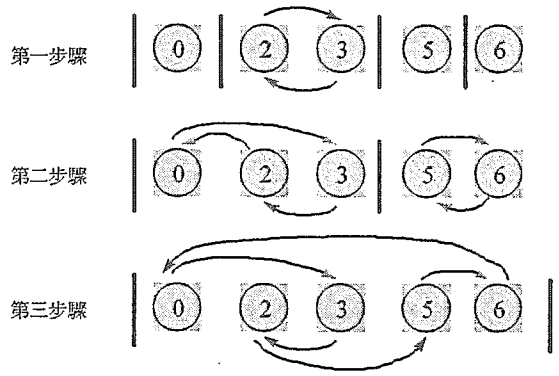

Fig. 5. {0,2,3,5,6}節點之環的連接過程

4.2 定理證明

以上是由例子來驗證我們所造出的環狀次序，其路徑是互斥的。接著，我們就要用理論來證明這個演算法在合併環的動作時，都不會發生路徑衝突的情形。所以，分別要證明此演算法對 3 種情形所採取的動作。這 3 個動作如下：

(1)建立兩條路徑，將兩個節點相連成長度爲 2 的環。

(2)將單一節點合併到環之中。

(3)將兩個環合併成一個較大的環。

在做合併環的動作時，所使用的節點並不是隨便挑選的，而是如同第三節中，要連接兩個 buddy 子網路的路徑的方法相同，要選擇兩個位址有最長相同字尾的節點才行。因此，上面(3)的動作，其實已經由定理五證明了結果是互斥的。以下我們就證明(1)和(2)的動作。

定理六

在 NxN 的 Omega 網路中，$L=\log_2 N$。若 A 和 B 是網路中的兩個節點，

則 Path(A,B)和 Path(B,A)互斥。

證明：

因爲 A 和 B 是相異節點，令 $Ast(A \downarrow B)=k$，$k \neq 0$。則

$$Ast(B \uparrow A)= Ast(A \uparrow B) > L-k$$

所以 $Ast(A \downarrow B)+Ast(B \uparrow A) > L$。

由定理一可知，Path(A,B)和 Path(B,A)互斥。得證。

定理七

假設在 NxN 的 Omega 網路之中，$L=\log_2 N$。令 C_1 和 C_2 是兩個 k 位元的 buddy 子網路，Path(A,B)和 Path(E,F)是 C_1 中的兩條互斥路徑，C 是 C_2 中的一個節點，且 $Ast(C \downarrow E) \geq Ast(C \downarrow A)$。則 Path(A,C)、Path(C,B) 和 Path(E,F)皆互斥。

證明：

此證明分成三部分

(1) 證明 Path(A,C)和 Path(E,F)互斥

因爲 Path(A,B)和 Path(E,F)互斥，

$$Ast(A \downarrow E)+Ast(B \uparrow F) > L。$$

又因爲 B、F 是 C_1 中的節點，而 C 是 C_2 中的節點，

所以 $Ast(C \uparrow F) > Ast(B \uparrow F)$。

故

$$Ast(A \downarrow E)+Ast(C \uparrow F) > Ast(A \downarrow E)+Ast(B \uparrow F) > L$$

即 Path(A,C)和 Path(E,F)互斥。

(2) 證明 Path(C,B)和 Path(E,F)互斥

Path(A,B)和 Path(E,F)互斥，且 $Ast(C \downarrow E) \geq Ast(C \downarrow A)$。由定理四，Path(C,B)和 Path(E,F)互斥。

(3)證明 Path(A,C)和 Path(C,B)互斥

因爲 A、B 是 C_1 中的節點，而 C 是 C_2 中的節點，

所以 $Ast(A \downarrow C) > L-k$，$Ast(C \uparrow B)=k+1$。

$$Ast(A \downarrow C)+ Ast(C \uparrow B) > (L-k)+(k+1) = L+1$$

即 Path(A,C)和 Path(C,B)互斥。

由以上的證明得知，Path(A,C)、Path(C,B) 和 Path(E,F)皆互斥。得證。

5. 結論

我們提出的這個演算法，可以將多階層網路上任意選擇的節點，安排一個環狀次序。依照著這個次序，這些處理器節點可以同時傳送資料而不會導致衝突。這對 group multicast 這類的通訊動作確實十分有效，這使得節點傳送資料的時間重疊，於是可以實現管線方式的效果，讓 N 個要做 group multicast 的處理器節點在 N-1 次的資訊傳遞便可以完成工作。

我們導入 buddy 子網路的觀念，因爲這個演算法是由最小的環，也就是長度爲 2 的環開始做起，再和其 buddy 子網路的環合併成較大的環。由最小的環開始，不斷地重複合併的動作，直到所有被任意選擇的節點都包含在完成的環之中爲止。這種方法的確是較直覺且有系統的方法，而且由於 buddy 子網路的特性，我們可以保證在每一個合併的步驟，所完成的環中都沒有會發生衝突的路徑。這使得這些節點可以順利地同時做資訊傳遞的工作。

在可重排網路上可以完成任意的排列，但是大部分的多階層網路，都和 Omega 網路一樣是屬於 blocking network。這類網路只能完成小部份的排列，我們利用這個演算法，就可以找到被任意選擇節點合法的排列，以完成所需的工作。這類的應用在排序或是矩陣的運算上都是十分常見的。

參考文獻

[1] P. K. McKinley and D. F. Robinson, "Collective communication in wormhole-routed massively parallel computers", *Computer*, Dec. 1995, pp.39-50.

[2] L. M. Ni, P. K. McKinley, "A survey of wormhole routing techniques in direct networks", *Computer*, Feb. 1993, pp.62-76.

[3] Y. Tseng, D. K. Panda and T. Lai,"A trip-based multicasting model in wormhole-routed networks with virtual channels", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 2, Feb. 1996, pp.138-146.

[4] X. Lin and L. M. Ni,"Deadlock free multicast wormhole routing in multicomputrt networks", *Proc. 18th Int'l Symposium on Computer Architecture*, 1991. pp. 116-125.

[5] P. K. McKinley, H. Xu, A. H. Esfahanian and L. M. Ni, "Unicast-Based multicast communication in wormhole-routed networks", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 12, Dec. 1994, pp.1252-1265.

[6] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks", *IEEE Trans. on Computers*, Vol. C-36, No. 5, May. 1987, pp. 547-553.

[7] D. H. Lawire, "Access and alignment of data in an array processors", *IEEE Trans. on Computers*, Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.