在郵遞模式下通訊數最佳的平行前置計算

林彥君　　葉清松
國立台灣科技大學 電子工程研究所
yclin@et.ntust.edu.tw

摘要

分散記憶體式(distributed-memory)的多處理器電腦(multicomputer)要靠處理器之間訊息傳遞(message passing)來合作完成計算工作。將所有的處理器視爲完全連結(fully connected)，而可彼此直接互傳訊息是最新的趨勢，這個能力在IBM SP2, nCUBE 2, Intel Paragon, CM-5等電腦皆可看到。前置計算(prefix computation)是對n個輸入值$v_1, v_2, \ldots, v_n$與具結合性的二元運算⊕，求得n個前置值$v_1 \oplus v_2 \oplus \ldots \oplus v_i$，$1 \le i \le n$。前置計算的應用很廣泛。本論文針對前置計算問題，在完全連結平行電腦的多埠郵遞(multiport postal)通訊模式，推導出所需通訊步數的最小極限。我們也提出兩個通訊步數最佳的(optimal)前置計算演算法。

關鍵詞：平行前置計算，郵遞模式，多埠，通訊數最佳，分散記憶體式的多處理器電腦

1. 簡介

前置計算 (prefix computation) 乃是針對 n 個元素 $v_1, v_2, \ldots, v_n$，以及具有結合性的 (associative) 二元運算 (binary operation) ⊕，計算出 n 個前置項 (prefixes) $v_1 \oplus v_2 \oplus \ldots \oplus v_i$，$1 \le i \le n$，其中二元運算⊕可以簡單到只是一個布林運算 (boolean operation)，或者複雜到是一個浮點數矩陣相乘 (floating-point matrix multiplication)。例如，對 3, 6, 8 三個元素以及加法運算，前置計算是要算出 3, 9, 17。前置計算有許多的應用，例如：迴圈的平行化 (loop parallelization)，線性遞迴之求解 (solution of linear recurrences)，進位前看加法器 (carry-look-ahead addition)，多項式的求值和插入法 (polynomial evaluation and interpolation)，以及數位濾波 (digital filtering) 等 [1, 10, 12, 14, 21, 23]。因爲前置計算的重要性，目前已有許多平行前置演算法 [10, 12, 13, 15, 16, 20] 以及平行前置電路 [9, 11, 13-15, 17-19, 22] 被設計出來解決前置計算的問題。

多處理器的平行電腦，依記憶體的存取方式，可分爲共用記憶體式 (shared-memory) 以及分散記憶體式 (distributed-memory) 的平行電腦。共用記憶體是指所有的處理器均存取一個相同的記憶體空間；而分散記憶體則是每個處理器都擁有屬於自己的記憶體空間。在共用記憶體式的平行電腦上，當處理器的數量愈多，存取記憶體時發生競爭的機率也相對提高，所以這一類平行電腦的處理器數量通常不會很多。在分散記憶體式的平行電腦上，處理器之間可以透過連結網路 (interconnection network)，以訊息傳遞的方式來傳送資料到其他的處理器。有許多種處理器的連結方式可以連結大量的處理器個數，例如 mesh, ring, torus, hypercube, multistage interconnection network 等。將所有的處理器視爲完全連結 (fully connected)，而可彼此直接互傳訊息是最新的趨勢；這個能力在 IBM SP2, nCUBE 2, Intel Paragon, CM-5 等電腦皆可看到 [2]。

在完全連結的平行電腦上，任何一對處理器之間的通訊距離均視爲相等。由於這種模式完全不需考慮處理器的眞正連結方式，所以可能反而有彈性 (flexible)；例如，在可以動態地 (dynamically) 分配處理器，或可容許處理器發生錯誤 (fault-tolerant) 的電腦下，演算法會更有可攜性 (portable) [6]。

在分散記憶體式的多處理器電腦 (multicomputer) 中，處理器之間要靠訊息傳遞 (message passing) 來合作完成計算工作。這種處理器之間的通訊，因著能力的不同又有各種不同的模式。例如，送或收 (send-or-receive) 模式是指每個處理器在同一時間只能送出或接收一組資料；而 1-埠 (1-port) 模式，或叫送且收 (send-and-receive) 模式，能同時對兩個不同的對象分別送出一組資料與接收一組資料；在雙向 (bidirectional) 模式，任一處理器皆可同時送與收，但送與收的對象必須是同一個處理器。

多埠 (multiport) 訊息傳遞模式 (簡稱爲多埠模式) 的每個處理器具有 k 個通訊埠，使得處理器在每一個通訊步驟中，能夠傳送 k 組資料到 k 個處理器，同時也能夠從 k 個不同的處理器共接收 k 組資料，故又稱爲 k-埠模式，其中 $k \ge 1$ [3, 6]。

在上述的數種模式中，若再考慮郵遞 (postal) 模式 [4, 5, 7] 的特性，則模式的種類又更增加。在郵遞模式中，第 j 步送出的資料，在第 $j+\lambda-1$ 步才會被收到，接收資料的處理器在第 $j+\lambda$ 步才能將該筆資料再傳送到其他處理器，其中 $\lambda \ge 1$。當 $\lambda = 1$，相當於不考慮郵遞模式。例如，k-埠郵遞模式具有 k-埠模式的特性，每個處理器有 k 個通訊埠，在同一個通訊步驟中，能夠傳送資料到最多 k 個處理器，並從 k 個不同的處理器共接收 k 組資料。由於在第 j 個通訊步驟傳送出去的資料，接收端在第 $j+\lambda-1$ 個通訊步驟才會收到，所以，在第 j 個通訊步驟所接收的資料，是在第 $j-\lambda+1$ 個通訊步驟所送出的。例如，當 $\lambda = 3$ 時，在第三個通訊步驟收到的資料，是在第一個通訊

步驟所送出；在第三個通訊步驟送出的資料，接收端在第五個通訊步驟才會收到。

在完全連結訊息傳遞的平行電腦上，針對不同的模式以及不同的運算，已經有許多相關的研究。在送或收模式以及多埠模式，針對前置計算問題，已有高效率的平行前置計算演算法 [16, 20]。與前置計算相關的研究有：聊天 (gossiping) 通訊、全域結合 (global combine) 運算。在多埠模式，針對聊天通訊，有通訊步數最佳的演算法 [8]。在多埠郵遞模式，針對全域結合運算，也有通訊步數最佳的演算法 [2]。

本論文中，我們針對完全連結的平行電腦，在多埠郵遞模式上，設計通訊步數最佳的平行前置計算演算法。在第2節，我們推導出前置計算演算法在多埠郵遞模式，所需通訊步數的最小極限 (lower bound)。第3節提出通訊步數最佳的演算法A，使用 n 個處理器，對 n 筆資料做前置計算。在第4節，我們提出演算法B，使用 $p<n$ 個處理器，就能以最佳的通訊步數完成前置計算。第5節爲結論。

2. 通訊步數的最小極限

定理 1：在 k-埠郵遞模式下，使用 n 個處理器，則前置計算演算法所需要的最少通訊步數爲 $\min\{i \mid G(i) \geq n\}$，其中

$$G(j) = 1, \qquad 0 \leq j < \lambda;$$
$$G(j) = G(j-1) + kG(j-\lambda), \qquad \lambda \leq j.$$

證明：令 PE i 表示第 i 個處理器，且 $v(i)$ 爲 PE i 的初始資料，$0 \leq i < n$。由於前置計算是要使 PE i 得到 $v(0) \oplus v(1) \oplus \ldots \oplus v(i)$ 的計算結果，所以每一個處理器都要有 $v(0)$ 或 $v(0)$ 參與計算所得到的值。爲了簡化，我們將只說處理器要有 $v(0)$。令 $G(j)$ 表示在第 j 步後，有 $v(0)$ 的處理器個數上限。在第 j 步時，$0 \leq j < \lambda$，因爲沒有任何處理器接收到資料，只有 PE 0 具有 $v(0)$，所以

$$G(j) = 1, \qquad 0 \leq j < \lambda.$$

當 $j \geq \lambda$，我們用歸納法來證明在第 j 步之後，最多有 $G(j) = G(j-1) + kG(j-\lambda)$ 個處理器有 $v(0)$。當 $j = \lambda$，除了 k 個處理器能在第 λ 步收到 $v(0)$，只有 PE 0 有 $v(0)$，所以最多有 $1 + k = G(j-1) + kG(j-\lambda)$ 個處理器有 $v(0)$。亦即，$G(j) = G(j-1) + kG(j-\lambda)$。

假設在第 j 步後，最多 $G(j) = G(j-1) + kG(j-\lambda)$ 個處理器有 $v(0)$，$\lambda \leq j < r$。在第 r 步時，最多有 $kG(r-\lambda)$ 個處理器收到 $G(r-\lambda)$ 個處理器在第 $r-\lambda+1$ 步時送出的 $v(0)$。所以 $G(r) = G(r-1) + kG(r-\lambda)$。

若前置計算要在第 i 步完成，則必須 $G(i) \geq n$，所以前置計算所需要的最少通訊步數爲 $\min\{i \mid G(i) \geq n\}$。 Q.E.D.

3. 前置計算演算法A

3.1 節介紹演算法的基本觀念。3.2 節提出演算法A與證明，並以一個例子說明執行過程與結果。

3.1. 演算法的基本觀念

我們要設計通訊步數最佳的演算法，首先要考慮如何將 $v(0)$ 傳送給其他處理器。由定理1的證明過程可知，在第 $j+\lambda-2 \geq 0$ 步之後，最多有 $G(j+\lambda-2)$ 個處理器擁有 $v(0)$；在第 $j+\lambda-1$ 步之後，最多有 $G(j+\lambda-1) = G(j+\lambda-2) + kG(j-1)$ 個處理器擁有 $v(0)$。亦即，通訊步數最佳的演算法必須有 $kG(j-1)$ 個處理器在第 $j+\lambda-1$ 步收到第 j 步所傳來，具有 $v(0)$ 的資料。接下來，要提出一個可能的資料傳送次序。

已知在第 $j-1 \geq 0$ 步後，總共有 $G(j-1)$ 個處理器擁有 $v(0)$ 這筆資料，且假設這 $G(j-1)$ 個處理器分別爲 PE 0, PE 1,..., PE $G(j-1)-1$；在第 $j+\lambda-2$ 步之後，總共有 $G(j+\lambda-2)$ 個處理器擁有 $v(0)$，且假設這 $G(j+\lambda-2)$ 個處理器分別爲 PE 0, PE 1,..., PE $G(j+\lambda-2)-1$。而且，在第 j 步，對所有的 $t \in \{0, 1, \ldots, k-1\}$，

PE 0 將 $v(0)$ 傳給 PE $G(j+\lambda-2) + tG(j-1)$，
PE 1 將 $v(0)$ 傳給 PE $1 + G(j+\lambda-2) + tG(j-1)$，
...，
PE $G(j-1)-1$ 將 $v(0)$ 傳給 PE $G(j-1) - 1 + G(j+\lambda-2) + tG(j-1)$。

也就是對所有 $t \in \{0, 1, \ldots, k-1\}$，

PE i 將 $v(0)$ 傳給 PE $i + G(j+\lambda-2) + tG(j-1)$，$0 \leq i \leq G(j-1)-1$，

因此，在第 $j+\lambda-1$ 步，會有 $kG(j-1)$ 個處理器收到 $v(0)$，所以總共有 $G(j+\lambda-2) + kG(j-1) = G(j+\lambda-1)$ 個處理器擁有 $v(0)$ 這筆資料。

我們用生成樹 (spanning tree) 來描述演算法中資料傳遞的情形。生成樹中每個節點 (node) 均有一個標示 (label) i，$0 \leq i \leq n-1$。若一個節點的標示爲 i，稱此節點爲節點 i。令 T_i 表示以節點 i 爲根節點的生成樹，且 T_i 包含節點 i 到節點 $n-1$，共 $n-i$ 個節點。T_i 最初只有節點 i，然後在建構步驟中加入節點，形成節點數較多的生成樹。令 x 表示第 $j-1 \geq 0$ 步後生成樹中的一個節點，我們以 $U_{x,j}$ 表示在第 j 個建構步驟中，所有新加入成爲節點 x 的子節點的集合：

$$U_{x,j} = \{y \mid y = x + G(j+\lambda-2) + tG(j-1) < n,\ 0 \leq t \leq k-1 \text{ 且 } t \text{ 爲整數}\}。$$

當 $y \in U_{x,j}$，表示 PE x 在第 j 步送出資料給 PE y。在郵遞模式中，第 j 步送出的資料在第 $j+\lambda-1$ 步才能被收到，接收資料的處理器在

第$j+\lambda$步才能將該筆資料再傳送到其他處理器。因此，在生成樹T_i的建構過程中，在第j步加入的子節點，在第$j+\lambda$步才能夠再增加新的子節點。例如當$n=10$，$k=2$，$\lambda=3$的情況下，$G(0)=1$, $G(1)=1$, $G(2)=1$, $G(3)=3$, $G(4)=5$, $G(5)=7$, $G(6)=13$，生成樹T_0的建構過程(圖1)如下：

步驟1 ($j=1$)：$U_{0,1}=\{1,2\}$，節點0加入子節點1, 2。

步驟2 ($j=2$)：$U_{0,2}=\{3,4\}$，節點0加入子節點3, 4。

步驟3 ($j=3$)：$U_{0,3}=\{5,6\}$，節點0加入子節點5, 6。

步驟4 ($j=4$)：$U_{0,4}=\{7\}$，節點0加入子節點7，
$U_{1,4}=\{8\}$，節點1加入子節點8，
$U_{2,4}=\{9\}$，節點2加入子節點9。

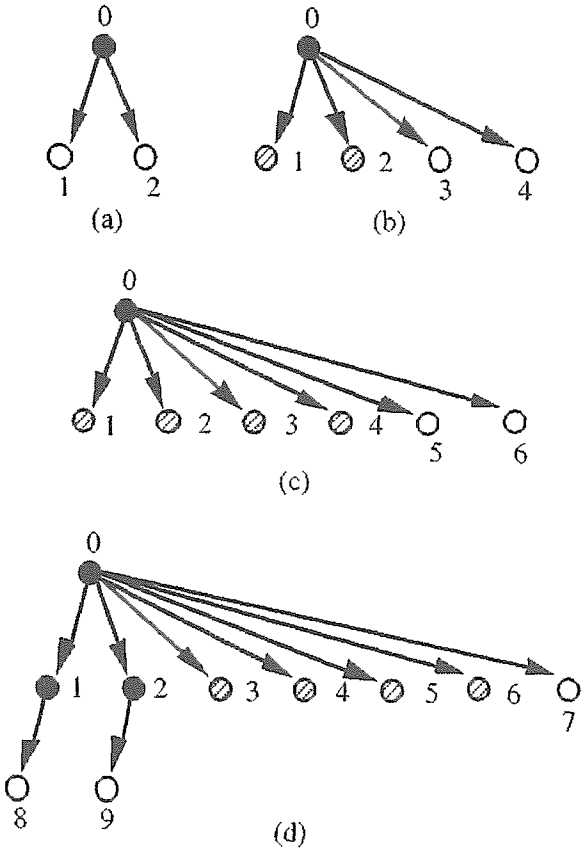

圖1. 在2-埠郵遞模式下建構生成樹T_0，$n=10$，$\lambda=3$，(a)步驟1 (b)步驟2 (c)步驟3 (d)步驟4。

當$j\geq 5$，$U_{x,j}=\phi$，$0\leq x\leq 9$，也就是在步驟4之後，生成樹中的任一個節點都不會再增加任何子節點。圖1中的實心黑點，表示此節點可以增加新的子節點；空心的節點代表在該建構步驟中增加的節點，這些節點在$\lambda-1$步之後才能夠再加入新的子節點。若將生成樹的建構步驟對應到演算法，則在第四步傳送給子節點的資料，在第六步才會收到。因爲步驟5與步驟6並沒有加入新的節點，所以在圖1中沒有畫出來。圖2顯示建構完成的生成樹T_1與T_2。

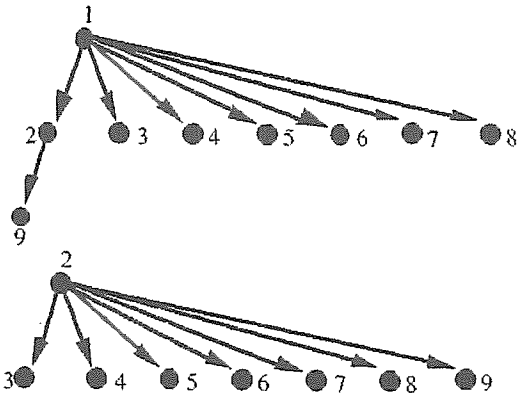

圖2. 在2-埠郵遞模式下建構完成的生成樹T_1與T_2，$n=10$。

3.2. 演算法A

假設有n個處理器，$v(i)$表示PE i的初始資料，$0\leq i<n$。由生成樹T_i的建構過程，我們可以推導出一個在k-埠郵遞模式下的前置計算演算法。我們將生成樹中的節點對應到處理器，生成樹T_i的建構步驟則相當於將$v(i)$傳送到其他處理器的步驟。每一個處理器在同一個步驟中，將接收到的全部資料都做$\oplus$運算，成爲一筆資料，所以在全部的通訊中，都只需傳送一筆資料給最多k個處理器。另外，因爲運算$\oplus$不一定具有交換性，所以必須確認每一筆資料的來源處理器，以便依序做計算。若$y\in U_{x,j-\lambda+1}$，即PE y在第j步，會從PE x收到在第$j-\lambda+1$步傳來的資料，則從$U_{x,j-\lambda+1}$的定義可知$y=x+G(j-1)+tG(j-\lambda)$，$0\leq t\leq k-1$。亦即，$x=y-G(j-1)-tG(j-\lambda)$，所以我們定義集合

$$W_{y,j}=\{x \mid x=y-G(j-1)-tG(j-\lambda)\geq 0,\ j\geq\lambda,\ 0\leq t\leq k-1 \text{且} t \text{爲整數}\},$$

表示PE y在第j步時，要從PE x，$x\in W_{y,j}$，收到第$j-\lambda+1$步傳來的資料。

圖3爲在PE i上執行的演算法，$0\leq i<n$，且演算法需要$\min\{j \mid G(j)\geq n\}$個通訊步數。

演算法 A. /*在k-埠郵遞模式，於PE i執行，解決前置計算問題的演算法，其中$0 \le i < n$。h表示集合$W_{y,j}$中元素的個數，且$0 \le h \le k$，e_r則表示集合$W_{y,j}$中，第r個最小的元素。每個PE i需要一個記憶體位置c，用來存放初始值，並且可用來儲存計算的中間值，以及最後的結果。在演算法中，我們以$c(i)$表示在PE i上的記憶體c。*/

$m := \min\{j \mid G(j) \ge n\}$
for $j = 1$ **to** m **do**
  if $1 \le j \le m - \lambda + 1$ **then**
    for all $y \in U_{i,j}$, send $c(i)$ to PE y
  end if
  if $\lambda \le j \le m$ **then**
    for all $r \in \{1, 2, \ldots, h\}$, receive $c(e_r)$ from PE e_r
    $c(i) := c(e_1) \oplus c(e_2) \oplus \ldots \oplus c(e_h) \oplus c(i)$
  end if
end for

圖 3. k-埠郵遞模式上的前置計算演算法。

我們以$n = 10$，$k = 2$，$\lambda = 3$的模式，使用10個處理器爲例，說明演算法的執行過程。因爲$G(5) = 7$, $G(6) = 13$，所以$m = 6$，也就是需要6個步驟。我們以$C_j = \langle z(0), z(1), \ldots, z(9)\rangle$表示PE 0, PE 1,..., PE 9在第$j$步之後，$c(0)$, $c(1)$,..., $c(9)$的值分別爲$z(0)$, $z(1)$,..., $z(9)$，其中$1 \le j \le 6$。C_0則代表每個處理器中$c(i)$的初始值，且令$c(i) = i$，即

$C_0 = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\rangle$.

在第一步，對於$0 \le i < 8$，$U_{i,1} = \{i + 1, i + 2\}$，而$U_{8,1} = \{9\}$，傳送的資料爲$C_0$。因爲沒有處理器收到任何資料，所以在第一步之後，

$C_1 = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\rangle$.

在第二步，對於$0 \le i < 6$，$U_{i,2} = \{i + 3, i + 4\}$，而$U_{6,2} = \{9\}$，傳送的資料爲$C_1$。因爲沒有處理器收到任何資料，所以在第二步之後，

$C_2 = \langle 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\rangle$.

在第三步，對於$0 \le i < 4$，$U_{i,3} = \{i + 5, i + 6\}$，而$U_{4,3} = \{9\}$，傳送的資料爲$C_2$。$W_{0,3} = \phi$，$W_{1,3} = \{0\}$，而對於$2 \le i < 10$，$W_{i,3} = \{i - 2, i - 1\}$。收到的資料爲第一步傳來的$C_0$，所以在第三步之後，

$C_3 = \langle 0, 0{:}1, 0{:}2, 1{:}3, 2{:}4, 3{:}5, 4{:}6, 5{:}7, 6{:}8, 7{:}9\rangle$.

在第四步，對於$0 \le i < 3$，$U_{i,4} = \{i + 7\}$，傳送的資料爲C_3。對於$0 \le i < 3$，$W_{i,4} = \phi$，$W_{3,4} = \{0\}$，而對於$4 \le i < 10$，$W_{i,4} = \{i - 4, i - 3\}$。收到的資料爲第二步傳來的$C_1$，所以在第四步之後，

$C_4 = \langle 0, 0{:}1, 0{:}2, 0{:}3, 0{:}4, 1{:}5, 2{:}6, 3{:}7, 4{:}8, 5{:}9\rangle$.

在第五步，處理器不會送出任何資料。對於$0 \le i < 5$，$W_{i,5} = \phi$，$W_{5,5} = \{0\}$，而對於$6 \le i < 10$，$W_{i,5} = \{i - 6, i - 5\}$。收到的資料爲第三步傳來的$C_2$，所以在第五步之後，

$C_5 = \langle 0, 0{:}1, 0{:}2, 0{:}3, 0{:}4, 0{:}5, 0{:}6, 1{:}7, 2{:}8, 3{:}9\rangle$.

在第六步，處理器不會送出任何資料。對於$0 \le i < 7$，$W_{i,6} = \phi$，而對於$7 \le i < 10$，$W_{i,6} = \{i - 7\}$。收到的資料爲第四步傳來的C_3，所以在第六步之後，

$C_6 = \langle 0, 0{:}1, 0{:}2, 0{:}3, 0{:}4, 0{:}5, 0{:}6, 0{:}7, 0{:}8, 0{:}9\rangle$.

所以演算法在第六步之後完成前置計算。

預備定理 2：在演算法A的第j ($\ge \lambda$)步，

情況 1.若$0 \le i < G(j - 1)$，則PE i不會收到任何資料；

情況 2.若$G(j - 1) + sG(j - \lambda) \le i < G(j - 1) + (s + 1)G(j - \lambda)$，$0 \le s \le k - 1$且$s$爲整數，則對所有的$u \in \{0, 1, \ldots, s\}$，PE i會由PE $(i - G(j - 1) - uG(j - \lambda))$收到$c(i - G(j - 1) - uG(j - \lambda))$；

情況 3.若$G(j - 1) + kG(j - \lambda) \le i < n$，則對所有的$t \in \{0, 1, \ldots, k - 1\}$，PE i會由PE $(i - G(j - 1) + tG(j - \lambda))$收到$c(i - G(j - 1) + tG(j - \lambda))$。

證明：我們會用到$W_{y,j} = \{x \mid x = y - G(j - 1) - tG(j - \lambda) \ge 0$，$j \ge \lambda$，$0 \le t \le k - 1$且$t$爲整數$\}$。

情況 1.$0 \le i < G(j - 1)$.

因爲$i < G(j - 1)$，由$W_{i,j} = \phi$可知PE i在第j步不會收到任何資料。

情況 2.$G(j - 1) + sG(j - \lambda) \le i < G(j - 1) + (s + 1)G(j - \lambda)$，$0 \le s \le k - 1$.

因爲$G(j - 1) + sG(j - \lambda) \le i < G(j - 1) + (s + 1)G(j - \lambda)$，所以$W_{i,j} = \{i - G(j - 1) - uG(j - \lambda) \mid u$爲整數且$0 \le u \le s\}$。故在第$j$步，對所有的$u \in \{0, 1, \ldots, s\}$，PE i會由PE $(i - G(j - 1) - uG(j - \lambda))$收到$c(i - G(j - 1) - uG(j - \lambda))$。

情況 3.$G(j - 1) + kG(j - \lambda) \le i < n$.

因爲$G(j - 1) + kG(j - \lambda) \le i$，故$W_{i,j} = \{i - G(j - 1) - tG(j - \lambda) \mid t$爲整數且$0 \le t \le k - 1\}$。所以在第$j$步，對所有的$t \in \{0, 1, \ldots, k - 1\}$，PE i會由PE $(i - G(j - 1) - tG(j - \lambda))$收到$c(i - G(j - 1) - tG(j - \lambda))$。 Q.E.D.

預備定理 3：演算法 A 在第 j (≥ 1) 步之後，PE i 有

$c(i) = 0{:}i, \qquad 0 \leq i < G(j);$

$c(i) = (i - G(j) + 1){:}i, \qquad G(j) \leq i < n.$

證明：我們的證明方式爲對 j 做歸納法。當 $1 \leq j < \lambda$，$G(j) = 1$。對於 $0 \leq i < n$，PE i 只有將初始值 $v(i)$ 傳送到 PE y，$y \in U_{i,j}$，還沒有任何處理器收到資料。所以，當 $1 \leq j < \lambda$ 時，$c(0) = v(0)$，且對於 $G(j) = 1 \leq i < n$，$c(i) = v(i)$，故當 $1 \leq j < \lambda$ 時，本定理成立。

接著，假設當 $j \leq T$ 時，本定理成立。由歸納法的假設可知，在第 $T - \lambda + 1$ 步之後，PE i 有

$c(i) = 0{:}i, \qquad 0 \leq i < G(T - \lambda + 1);$

$c(i) = (i - G(T - \lambda + 1) + 1){:}i, \qquad G(T - \lambda + 1) \leq i < n.$

在第 T 步之後，PE i 有

$c(i) = 0{:}i, \qquad 0 \leq i < G(T);$

$c(i) = (i - G(T) + 1){:}i, \qquad G(T) \leq i < n.$

當 $j = T + 1$ 時，處理器會收到第 $T - \lambda + 2$ 步所送出的資料，也就是在第 $T - \lambda + 1$ 步之後的計算結果，並與第 T 步所得的結果做計算。我們將 i 的範圍分成兩個情況來討論處理器在第 $T + 1$ 步的計算情形：

情況 1. $0 \leq i < G(T + 1)$.

當 $G(T) \leq i < G(T + 1)$，則 $G(T) \leq i < n$，故由歸納法的假設可知，PE i 在第 T 步之後有

$$c(i) = (i - G(T) + 1){:}i.$$

接著，檢查 PE i 在第 $T + 1$ 步可收到什麼資料。下列 i 的 k 個範圍，我們針對任意一個範圍來說明：$G(T) \leq i < G(T) + G(T - \lambda + 1)$, $G(T) + G(T - \lambda + 1) \leq i < G(T) + 2G(T - \lambda + 1)$,..., $G(T) + (k - 1)G(T - \lambda + 1) \leq i < G(T) + kG(T - \lambda + 1) = G(T + 1)$。

當 $G(T) + sG(T - \lambda + 1) \leq i < G(T) + (s + 1)G(T - \lambda + 1)$ 時，其中 $0 \leq s \leq k - 1$，由預備定理 2 可知，在第 $T + 1$ 步，對所有的 $u \in \{0, 1,..., s\}$，PE i 會由處理器 PE $(i - G(T) - uG(T - \lambda + 1))$ 收到 $c(i - G(T) - uG(T - \lambda + 1))$，一共收到 $s + 1$ 筆資料。接下來，我們說明這 $s + 1$ 筆資料的內容。

因爲 $G(T) + sG(T - \lambda + 1) \leq i < G(T) + (s + 1)G(T - \lambda + 1)$，故 $0 \leq i - G(T) - sG(T - \lambda + 1) < G(T - \lambda + 1)$。所以，從歸納法的假設可知，在第 $T - \lambda + 1$ 步之後，PE $(i - G(T) - sG(T - \lambda + 1))$ 有

$$c(i - G(T) - sG(T - \lambda + 1)) = 0{:}(i - G(T) - sG(T - \lambda + 1)). \tag{1}$$

因爲 $G(T) + sG(T - \lambda + 1) \leq i$，故 $G(T - \lambda + 1) \leq i - G(T) - (s - 1)G(T - \lambda + 1)$。因此，對所有的 $w \in \{0, 1,..., s - 1\}$，$G(T - \lambda + 1) \leq i - G(T) - wG(T - \lambda + 1)$。又因爲 $i < n$，所以 $G(T - \lambda + 1) \leq i - G(T) - wG(T - \lambda + 1) < n$。從歸納法的假設可知，在第 $T - \lambda + 1$ 步之後，PE $(i - G(T) - wG(T - \lambda + 1))$ 有

$$c(i - G(T) - wG(T - \lambda + 1)) = (i - G(T) - (w + 1)G(T - \lambda + 1) + 1){:}(i - G(T) - wG(T - \lambda + 1)). \tag{2}$$

式 (1) 爲 PE i 在第 $T + 1$ 步所收到的一筆資料，式 (2) 則表示另外的 s 筆資料。在第 $T + 1$ 步，PE i 將收到的全部資料與式 (2) 的 $c(i)$ 做運算，即 $c(i) := c(i - G(T) - sG(T - \lambda + 1)) \oplus c(i - G(T) - (s - 1)G(T - \lambda + 1)) \oplus \ldots \oplus c(i - G(T)) \oplus c(i)$，也就是

$$\begin{aligned} c(i) := {} & [0{:}(i - G(T) - sG(T - \lambda + 1))] \\ & \oplus [(i - G(T) - sG(T - \lambda + 1) + 1){:}(i - G(T) - (s - 1)G(T - \lambda + 1))] \\ & \oplus [(i - G(T) - (s - 1)G(T - \lambda + 1) + 1){:}(i - G(T) - (s - 2)G(T - \lambda + 1))] \\ & \oplus \ldots \\ & \oplus [(i - G(T) - G(T - \lambda + 1) + 1){:}(i - G(T))] \\ & \oplus [(i - G(T) + 1){:}i]. \end{aligned}$$

所以，

$$c(i) = 0{:}i, \quad G(T) \leq i < G(T) + kG(T - \lambda + 1) = G(T + 1). \tag{3}$$

另外，當 $0 \leq i < G(T)$，由歸納法的假設可知，在第 T 步之後，

$$c(i) = 0{:}i, 0 \leq i < G(T). \tag{4}$$

由式 (3) 及式 (4) 可得

$c(i) = 0{:}i, \qquad 0 \leq i < G(T + 1).$

情況 2. $G(T + 1) \leq i < n$.

當 $G(T + 1) \leq i < n$，則由歸納法的假設可知，PE i 在第 T 步之後有

$$c(i) = (i - G(T) + 1){:}i. \tag{5}$$

接著，檢查 PE i 在第 $T + 1$ 步可收到什麼資料。由於 $G(T + 1) \leq i < n$，由預備定理 2 可知，在第 $T + 1$ 步，對所有的 $t \in \{0, 1,..., k - 1\}$，PE i 會由 k 個處理器 PE $(i - G(T) - tG(T - \lambda + 1))$ 收到 $c(i - G(T) - tG(T - \lambda + 1))$，一共收到 k 筆資料。接下來，我們說明這 k 筆資料的內容。

因爲 $G(T + 1) = G(T) + kG(T - \lambda + 1) \leq i$，故 $G(T - \lambda + 1) \leq i - G(T) - (k - 1)G(T - \lambda + 1)$。因此，對所有的 $t \in \{0, 1,..., k - 1\}$，$G(T - \lambda + 1) \leq i - G(T) - tG(T - \lambda + 1)$。又因爲 $i < n$，所以 $G(T - \lambda + 1) \leq i - G(T) - tG(T - \lambda + 1) < n$。從歸納法的假設可知，在第 $T - \lambda + 1$ 步之後，PE $(i - G(T) - tG(T - \lambda + 1))$ 有

$$c(i - G(T) - tG(T - \lambda + 1)) = (i - G(T) - (t + 1)G(T - \lambda + 1) + 1){:}(i - G(T) - tG(T - \lambda + 1)). \tag{6}$$

式 (6) 表示 PE i 在第 $T + 1$ 步所收到的 k 筆資料。PE i 將全部資料與式 (5) 的 $c(i)$ 做運算，即 $c(i) := c(i - G(T) - (k - 1)G(T - \lambda + 1)) \oplus c(i - G(T) - (k - 2)G(T - \lambda + 1)) \oplus \ldots \oplus c(i -$

$G(T)) \oplus c(i)$，也就是

$$
\begin{aligned}
c(i) := {} & [(i - G(T) - kG(T - \lambda + 1) + 1):(i - \\
& G(T) - (k-1)G(T-\lambda+1))] \\
\oplus {} & [(i - G(T) - (k-1)G(T - \lambda + 1) + 1):(i - \\
& G(T) - (k-2)G(T-\lambda+1))] \\
\oplus {} & \ldots \\
\oplus {} & [(i - G(T) - G(T-\lambda+1) + 1):(i - G(T))] \\
\oplus {} & [(i - G(T) + 1):i].
\end{aligned}
$$

所以，

$$
\begin{aligned}
c(i) &= (i - G(T) - kG(T-\lambda+1) + 1):i \\
&= (i - G(T+1) + 1):i, \quad G(T+1) \le i < n.
\end{aligned}
$$

由情況1可知，當 $0 \le i < G(T+1)$ 時，$c(i) = 0:i$；而由情況2可知，當 $G(T+1) \le i < n$ 時，$c(i) = (i - G(T+1) + 1):i$。因此，當 $j = T + 1$ 時，本定理成立。 Q.E.D.

定理4：演算法A是正確的。

證明：由預備定理3可知，在第 j 步之後，若 $0 \le i < G(j)$，PE i 有 $c(i) = 0:i$。因爲 $m = \min\{j \mid G(j) \ge n\}$，故 $n \le G(j)$。所以，演算法A執行結束後，也就是執行了 m 步之後，PE i 有 $c(i) = 0:i$，$0 \le i < n$。 Q.E.D.

4. 使用較少處理器的演算法B

當處理器的個數小於資料的個數時，我們提出另一個演算法（圖4），使用 $p < n$ 個處理器對 n 筆資料做前置計算。令 $q = n/p$，PE 0 到 PE $(n - p \times \lfloor q \rfloor - 1)$ 各分配 $\lceil q \rceil$ 筆資料，其餘的處理器則各分配 $\lfloor q \rfloor$ 筆資料。例如，當 $n = 75$，$p = 10$，則PE 0到PE 4各分配8筆資料，PE 5到PE 9各分配7筆資料。爲了方便表示，我們假設 q 爲整數，則每個處理器分配到的 q 個初始值爲 $v(iq), v(iq+1), \ldots, v((i+1)q-1)$，$0 \le i < p$。需要的記憶體有 c、d、$temp$、陣列 v 及陣列 u。在演算法中，我們以 $c(i)$ 以及 $d(i)$ 分別表示在PE i 上的記憶體 c 與 d，陣列 $v[iq..(i+1)q-1]$ 用來存放 q 個初始值，陣列 $u[iq..(i+1)q-1]$ 則是用來儲存最後的計算結果 $0:iq, 0:iq+1, \ldots, 0:(i+1)q-1$。$c(i)$ 的初始值爲 $v(iq) \oplus v(iq+1) \oplus \ldots \oplus v((i+1)q-1)$，$d(i)$ 的初始值則爲 $v(iq)$，$temp$ 則是用來存放計算的中間值。

演算法B在執行時，各個處理器的傳送路徑與演算法A是一樣的。另外，$c(i)$ 與 $d(i)$ 所作的運算也是相同的，兩者所不同的只有初始值。故由預備定理3及定理4可知，在第 m 步之後，$c(i) = 0:(i+1)q-1$ 且 $d(i) = 0:iq$。故取 $u(iq) := d(i)$，並計算 $u(iq+j) = u(iq+j-1) \oplus v(iq+j)$，$1 \le j \le q-1$，則對於 $0 \le i < p$，所有的 PE i 均有 $u(iq) = 0:iq, u(iq+1) = 0:iq+1, \ldots, u((i+1)q-1) = 0:(i+1)q-1$ 等 q 個前置項的計算結果。

演算法B. /* 在 k-埠郵遞模式，於PE i 執行，使用 p 個處理器對 n 筆資料做前置計算的演算法，其中 $0 \le i < p < n$。演算法中 $q = n/p$，h 表示集合 $W_{x,j}$ 中元素的個數，且 $0 \le h \le k$，e_r 則表示集合 $W_{x,j}$ 中，第 r 個最小的元素。 */

```
c(i) := v(iq) ⊕ v(iq + 1) ⊕ ... ⊕ v((i + 1)q − 1)
d(i) := v(iq)
m := min {j | G(j) ≥ p}
for j = 1 to m do
  if 1 ≤ j ≤ m − λ + 1 then
    for all y ∈ U_{i,j}, send c(i) to PE y
  end if
  if λ ≤ j ≤ m then
    for all r ∈ {1, 2,..., h}, receive c(e_r) from PE e_r
    temp := c(e_1) ⊕ c(e_2) ⊕ ... ⊕ c(e_h)
    c(i) := temp ⊕ c(i)
    d(i) := temp ⊕ d(i)
  end if
end for
u(iq) := d(i)
for j = 1 to q − 1 do
  u(iq + j) := u(iq + j − 1) ⊕ v(iq + j)
end for
```

圖4. k-埠郵遞模式上使用 $p < n$ 個處理器的前置計算演算法。

我們以 $n = 80$，$k = 2$，$\lambda = 3$ 的模式，使用10個處理器爲例，說明演算法的執行過程。因爲 $G(5) = 7$, $G(6) = 13$，所以 $m = 6$，也就是需要六個步驟。我們以 $C_j = \langle z(0), z(1), \ldots, z(9) \rangle$ 表示 PE 0, PE 1,..., PE 9 在第 j 步之後，$c(0), c(1), \ldots, c(9)$ 的值分別爲 $z(0), z(1), \ldots, z(9)$。同樣的，我們以 $D_j = \langle y(0), y(1), \ldots, y(9) \rangle$ 表示 PE 0, PE 1,..., PE 9 在第 j 步之後，$d(0), d(1), \ldots, d(9)$ 的值分別爲 $y(0), y(1), \ldots, y(9)$，其中 $0 \le j \le 6$。C_0 及 D_0 則分別代表每個處理器中 $c(i)$ 與 $d(i)$ 的初始值，並直接以 i 表示 $v(i)$。

由於 $n = 80$，$p = 10$，故 $q = n/p = 8$，所以PE 0到PE 9各分配8筆資料，因此

C_0 = <0:7, 8:15, 16:23, 24:31, 32:39, 40:47, 48:55, 56:63, 64:71, 72:79>,

D_0 = <0, 8, 16, 24, 32, 40, 48, 56, 64, 72>.

在第一步，對於 $0 \le i < 8$，$U_{i,1} = \{i+1, i+2\}$，而 $U_{8,1} = \{9\}$，傳送的資料爲 C_0。因爲沒有收到任何資料，所以在第一步之後，

C_1 = <0:7, 8:15, 16:23, 24:31, 32:39, 40:47, 48:55, 56:63, 64:71, 72:79>,

D_1 = <0, 8, 16, 24, 32, 40, 48, 56, 64, 72>.

在第二步，對於 $0 \le i < 6$，$U_{i,2} = \{i+3, i+4\}$，而 $U_{6,2} = \{9\}$，傳送的資料爲 C_1。因爲沒有收到任何資料，所以在第二步之後，

C_2 = <0:7, 8:15, 16:23, 24:31, 32:39, 40:47, 48:55, 56:63, 64:71, 72:79>,

D_2 = <0, 8, 16, 24, 32, 40, 48, 56, 64, 72>.

在第三步，對於 $0 \le i < 4$，$U_{i,3} = \{i+5, i+6\}$，而 $U_{4,3} = \{9\}$，傳送的資料爲 C_2。$W_{0,3} = \phi$，$W_{1,3} = \{0\}$，而對於 $2 \le i < 10$，$W_{i,3} = \{i-2, i-1\}$。收到的資料爲第一步傳來的 C_0，所以在第三步之後，

C_3 = <0:7, 0:15, 0:23, 8:31, 16:39, 24:47, 32:55, 40:63, 48:71, 56:79>,

D_3 = <0, 0:8, 0:16, 8:24, 16:32, 24:40, 32:48, 40:56, 48:64, 56:72>.

在第四步，對於 $0 \le i < 3$，$U_{i,4} = \{i+7\}$，傳送的資料爲 C_3。對於 $0 \le i < 3$，$W_{i,4} = \phi$，$W_{3,4} = \{0\}$，而對於 $4 \le i < 10$，$W_{i,4} = \{i-4, i-3\}$。收到的資料爲第二步傳來的 C_1，所以在第四步之後，

C_4 = <0:7, 0:15, 0:23, 0:31, 0:39, 8:47, 16:55, 24:63, 32:71, 40:79>,

D_4 = <0, 0:8, 0:16, 0:24, 0:32, 8:40, 16:48, 24:56, 32:64, 40:72>.

在第五步，處理器不會送出任何資料。對於 $0 \le i < 5$，$W_{i,5} = \phi$，$W_{5,5} = \{0\}$，而對於 $6 \le i < 10$，$W_{i,5} = \{i-6, i-5\}$。收到的資料爲第三步傳來的 C_2，所以在第五步之後，

C_5 = <0:7, 0:15, 0:23, 0:31, 0:39, 0:47, 0:55, 8:63, 16:71, 24:79>,

D_5 = <0, 0:8, 0:16, 0:24, 0:32, 0:40, 0:48, 8:56, 16:64, 24:72>.

在第六步，處理器不會送出任何資料。對於 $0 \le i < 7$，$W_{i,6} = \phi$，而對於 $7 \le i < 10$，$W_{i,6} = \{i-7\}$。收到的資料爲第四步傳來的 C_3，所以在第六步之後，

C_6 = <0:7, 0:15, 0:23, 0:31, 0:39, 0:47, 0:55, 0:63, 0:71, 0:79>,

D_6 = <0, 0:8, 0:16, 0:24, 0:32, 0:40, 0:48, 0:56, 0:64, 0:72>.

再設定 $u(8i) := d(i)$，$0 \le i < 10$，則 $u(8i) = 0{:}8i$。每個處理器再執行運算 $u(8i+j) = u(8i+j-1) \oplus v(8i+j)$，$1 \le j < 8$，即算出 $u(8i+1), u(8i+2), \ldots, u(8i+7)$，且 $u(8i+j) = 0{:}(8i+j)$。所以，演算法在6步之後完成。

5. 結論

本論文針對多埠郵遞模式，推導出前置計算所需通訊步數的最小極限。我們也提出兩個通訊步數最佳的平行前置計算演算法：演算法A使用 n 個處理器，能夠用最少的通訊步數完成前置計算；演算法B則使用 $p < n$ 個處理器，也能夠用最少的通訊步數完成前置計算。

國科會計畫編號：NSC 87-2213-E-011-005

參考資料

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] A. Bar-Noy, *et al.*, "Computing global combine operations in the multiport postal model," *IEEE Trans. Parallel Distributed Syst.*, vol. 6, pp. 896-900, Aug. 1995.

[3] A. Bar-Noy and C.T. Ho, "Broadcasting multiple messages in the multiport model," in *Proc. 10th Int. Parallel Processing Symp.*, 1996, pp. 781-788.

[4] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the postal model for message-passing systems," *Math. Systems Theory*, vol. 27, pp. 431-452, 1994.

[5] A. Bar-Noy and S. Kipnis, "Multiple message broadcasting in the postal model," *Networks*, vol. 29, pp. 1-10, Jan. 1997.

[6] J. Bruck and C.T. Ho, "Efficient global combine operations in multiport message-passing systems," *Parallel Process. Lett.*, vol. 3, pp. 335-346, Dec. 1993.

[7] J. Bruck and C.T. Ho, "On the design and implementation of broadcast and global combine operations using the postal model," *IEEE Trans. Parallel Distributed Syst.*, vol. 7, pp. 256-265, Mar. 1996.

[8] J. Bruck, *et al.*, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 8, pp. 1143-1155, Nov. 1997.

[9] D.A. Carlson and B. Sugla, "Limited width parallel prefix circuits," *J. Supercomput.*, vol. 4, pp. 107-129, June 1990.

[10] R. Cole and U. Vishkin, "Faster optimal parallel prefix sums and list ranking," *Infom. Contr.*, vol. 81, pp. 334-352, 1989.

[11] F.E. Fich, "New bounds for parallel prefix circuits," in *Proc. 15th Symp. on the Theory of Computing*, 1983, pp. 100-109.

[12] C.P. Kruskal, T. Madej, and L. Rudolph, "Parallel prefix on fully connected direct connection machines," in *Proc. Int. Conf. on Parallel Processing*, 1986, pp. 278-284.

[13] R.E. Ladner and M.J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, pp. 831-838, Oct. 1980.

[14] S. Lakshmivarahan and S.K. Dhall, *Parallel Computing Using the Prefix Problem*. Oxford, UK: Oxford University Press, 1994.

[15] Y.-C. Lin, "Optimal parallel prefix circuits with fan-out 2 and corresponding parallel algorithms,"

Neural, Parallel & Scientific Computations, vol. 7, pp. 33-42, Mar. 1999.

[16] Y.-C. Lin and C.M. Lin, "Efficient parallel prefix algorithms on multicomputers," to appear in *J. Information Science Engineering*, vol. 16, Jan. 2000.

[17] Y.-C. Lin and C.-K. Liu, "Finding optimal parallel prefix circuits with fan-out 2 in constant time," *Inform. Process. Lett.*, vol. 70, pp. 191-195, May 1999.

[18] Y.-C. Lin and C.-C. Shih, "A new class of depth-size optimal parallel prefix circuits," *J. Supercomput.*, vol. 14, pp. 39-52, July 1999.

[19] Y.-C. Lin and C.-C. Shih, "Optimal parallel prefix circuits with fan-out at most 4," in *Proc. 2nd IASTED Int. Conf. on Parallel and Distributed Computing and Networks*, 1998, pp. 312-317.

[20] Y.-C. Lin and C.-S. Yeh, "Efficient parallel prefix algorithms on multiport message-passing systems," *Inform. Process. Lett.*, vol. 71, pp. 91-95, July 1999.

[21] A. Nicolau and H. Wang, "Optimal schedule for parallel prefix computation with bounded resources," in *Proc. Third ACM SIGPLAN Symp. on Principles & Practice Parallel Programming*, 1991, pp. 1-10.

[22] M. Snir, "Depth-size trade-offs for parallel prefix computation," *J. Algorithms*, vol. 7, pp. 185-201, 1986.

[23] H. Wang, A. Nicolau, and K.S. Siu, "The strict time lower bound and optimal schedules for parallel prefix with resource constraints," *IEEE Trans. Comput.*, vol. 45, pp. 1257-1271, Nov. 1996.