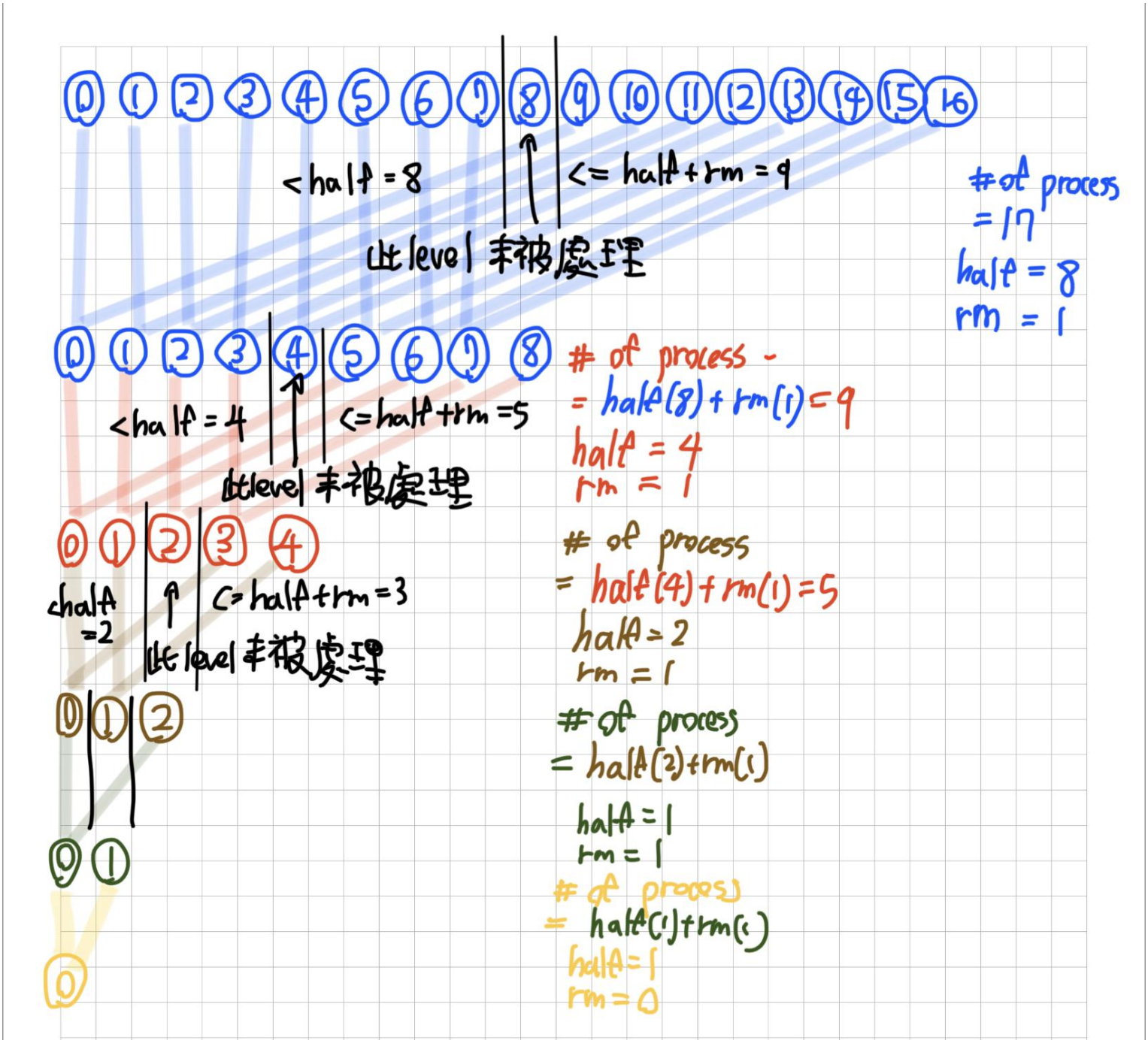
F74109016 葉惟欣 多處理機平行程式設計 HW1

第一題:

問題 1 : What have you done?

這一題我上網找了樹狀結構的平行處理傳遞方式。這種方式可以處理奇數個節點，他是用餘數的方式來判斷，如果在樹狀結構當前 level 要處理的 processor 的數量為奇數，那麼 my_rank id 為中間的節點在這個 level 下就不用做事，等待下一輪再傳遞做事。



最後因為 remain 為一就跳出 while 迴圈結束平行處理的階層處理。

每個 processor 都會跑這個迴圈而用自己的 my_rank 的值來判斷自己要做甚麼事情。像上圖直線就是不用傳遞而斜線就是從上的 processor 送資訊給 下的 processor。

送方:
比如 my_rank = 16 要做 send 的動作，MPI_SEND 此時也是符合 half(8)+rm(1) <= my_rank(16) && my_rank(16) < remain(17)。而我們在 MPI_SEND 的函數裡面也會計算 destination 的 processor 為 my_rank - half - rm 。 相當於平移 half + rm 的數字。這也可以說明為什麼如果為奇數個 processor 中間值得 processor 剛好不用做事，因為平移的數量(含餘數)剛好為奇數，而此時剛好 destination processor 到 processor 7。

收方:
My_rank = 7 的 processor 也符合 MPI_RECV 的 my_rank < half(8)。而我們在 MPI_SEND 的函數裡面也會計算 source processor 為 my_rank + (half + rm) 16。也因此接收兩端的溝通就會被建立了。

問題 2 : Analysis on your result

我比較**(1)普通的執行方法**跟**(2)平行用 broadcast 的傳遞**的和與**(3)平行用樹狀結構傳遞**的三種方法。

(1)普通的執行方法

```
F74109016@sivslab-pn1:~$ gcc problem1_sequential.c -o ./test
F74109016@sivslab-pn1:~$ ./test

0) 1001100111110101

0) 1001100111110110

0) 1001100111110111

0) 1001101111110101

0) 1001101111110110

0) 1001101111110111

0) 1001110111110101

0) 1001110111110110

0) 1001110111110111
finished in time 0.007032 secs.

A total of 9 solutions were found.
```

(2)平行用 broadcast 的傳遞

# of processor : 5

```
F74109016@sivslab-pn1:~$ mpicc problem1_parallel.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 5 ./test
0) 1001100111110111
0) 1001101111110101
4) 1001100111110110
4) 1001110111110111
1) 1001101111110110
2) 1001101111110111
3) 1001100111110101
3) 1001110111110110
2) 1001110111110101
Process 0 finished in time 0.000131 secs.
Process 0 finished.

A total of 9 solutions were found.
```

# of processor : 17

```
F74109016@sivslab-pn1:~$ mpicc problem1_parallel.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 17 ./test
7) 1001100111110101
9) 1001100111110111
9) 1001101111110101
11) 1001101111110111
11) 1001110111110101
13) 1001110111110111
8) 1001100111110110
10) 1001101111110110
12) 1001110111110110
Process 0 finished in time 0.026200 secs.
Process 0 finished.

A total of 9 solutions were found.
```

# of processor : 40

```
F74109016@sivslab-pn1:~$ mpicc problem1_parallel.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 40 ./test
13) 1001100111110101
6) 1001101111110110
37) 1001110111110101
7) 1001101111110111
38) 1001110111110110
15) 1001100111110111
14) 1001100111110110
5) 1001101111110101
Process 0 finished in time 0.010397 secs.
39) 1001110111110111
Process 0 finished.

A total of 9 solutions were found.
```

(3)平行用樹狀結構傳遞

# of processor : 5

```
F74109016@sivslab-pn1:~$ mpicc problem1.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 5 ./test

0) 1001100111110111

0) 1001101111110101

1) 1001101111110110

2) 1001101111110111

2) 1001110111110101

3) 1001100111110101

3) 1001110111110110

4) 1001100111110110

4) 1001110111110111
Process 0 finished in time 0.000417 secs.

A total of 9 solutions were found.
```

# of processor : 17

```
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 17 ./test

10) 1001101111110110

12) 1001110111110110

7) 1001100111110101

9) 1001100111110111

9) 1001101111110101

11) 1001101111110111

11) 1001110111110101

13) 1001110111110111

8) 1001100111110110
Process 0 finished in time 0.001690 secs.

A total of 9 solutions were found.
```

# of processor : 40

```
F74109016@sivslab-pn1:~$ mpicc problem1.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 40 ./test

37) 1001110111110101

38) 1001110111110110

39) 1001110111110111

6) 1001101111110110

7) 1001101111110111

13) 1001100111110101

14) 1001100111110110

15) 1001100111110111

5) 1001101111110101
Process 0 finished in time 0.010029 secs.

A total of 9 solutions were found.
```

分析:
普通的執行方法
用時
0.007032

平行用 broadcast 的傳遞
用時
# of p = 5 : 0.000131(第一名)
# of p = 17 : 0.0262
# of p = 40 : 0.010397

平行用樹狀結構傳遞
用時
# of p = 5: 0.0004 (第二名)
# of p = 17 : 0.001690
# of p = 40 : 0.010029

觀察 1:
平行處理只要 processor 數量較小就可以贏過普通循序。我認為可能是 processor 溝通也需要花時間，因此就算平行，如果 processor 的數量太多所耗的時間也不會比普通的處理方式還少。

觀察 2:
平行處理循序平行隨著 processor 變多，用甚麼方法來傳遞的差距越來越小。
Processor 數量 5 樹狀傳遞快 broadcast 傳遞 4 倍
Processor 數量 17 樹狀傳遞快 broadcast 傳遞 1.25 倍
Processor 數量 40 樹狀傳遞快 broadcast 傳遞 1...倍

問題 3 : Any difficulites?

這次的作業比較困難的部分有兩點:
第一點:
我原本沒有留意奇數與偶數的問題，所以上網找了答案，因為這件事讓我真正了解平行處理的傳遞方式。
後來我有跟同學討論，得到有三種解法，一種是我自己的解法，一種是直接一直找隔壁 partner 的解法，如果沒有隔避 partner 就下一回合再找，另一種跟我的解法很像但不過更聰明，他是直接用(m+1)/2 的算法去算的。
第二點:
是我原本就直接將答案的 range 除以 processor 的數量，所以我原本 processor 數字很大都沒有問題，但當數字很小有些答案就沒有被算到，算是粗心造成的失誤。

問題 4 : (optional) Feedback to TAs

作業很好玩，同樣是樹狀傳遞，每個同學都有不同的想法。

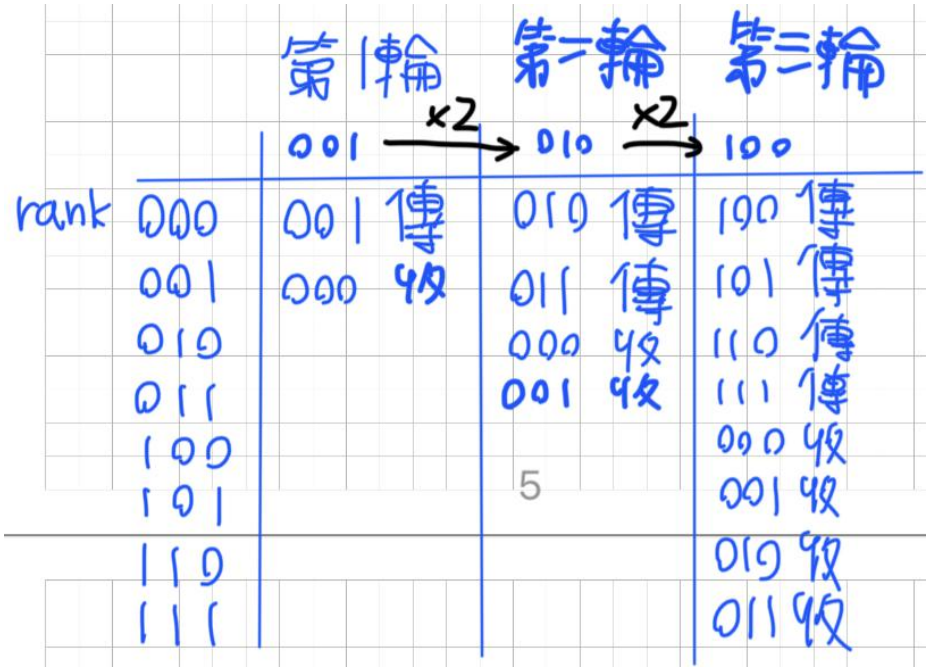
第二題:

問題 1 : What have you done?

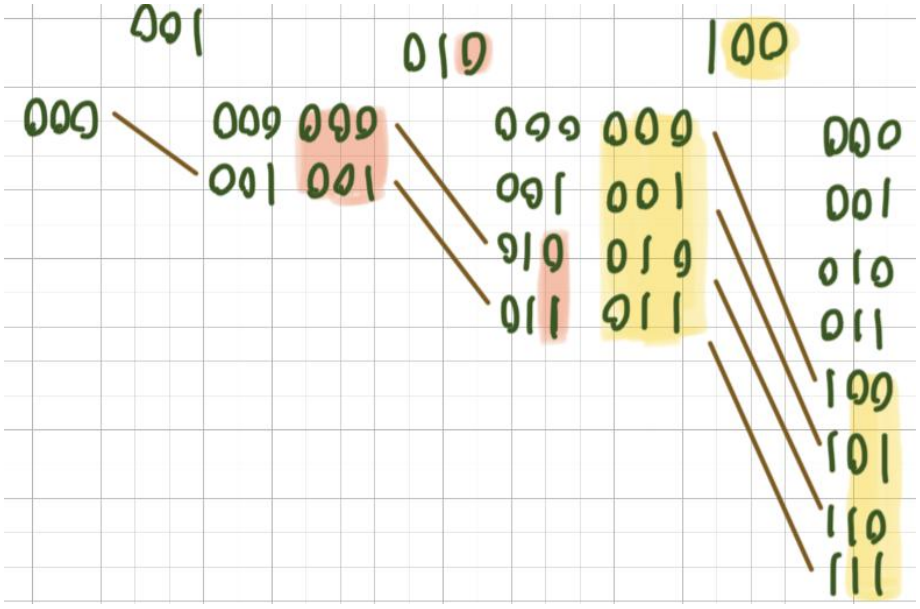
第二題其實跟第一題很像，傳遞方式都一樣，有不同的地方就是得到的結果又要再傳給每個 processor。

broadcast point to point 將 number_of_tosses 傳給每個 node。

因為我不想直接用內建的 MPI_bcast。所以我跟同學上網查了一個很厲害的做法。也就是用 bitmask。再不斷地參透 bitmask 的演算法後，我真的覺得這實在是太奇妙了。



他每一輪 bitmask 都會左移 1。做為遮罩。

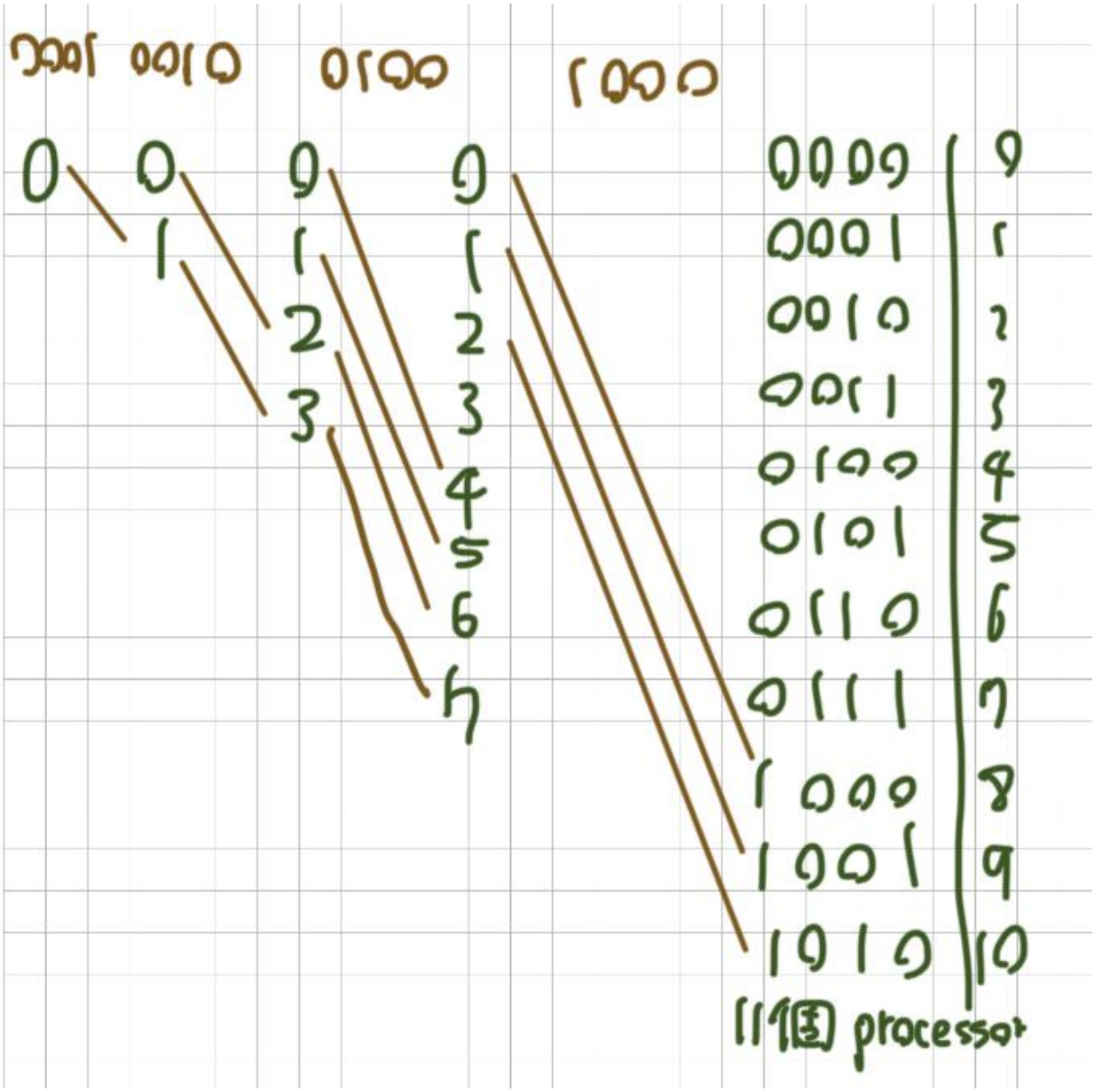


bitmask 做 exclusize or 相通為 0，相異為 1。如此一來可以將想要的傳給該點。像 100 101 110 111 都大於等於 100 所以為這次要處理的對象，而他們由誰傳遞給他們呢?應該說誰可以傳遞給他們，就是<100 的 rank 值(000 001 010 011)

的 processor。而如何分配就可以直接將他與 bitmask 100 做 exclusize or 就會得到後兩個 bit，剛好對應給傳遞他們的 processor (000 001 010 011)。

而為什麼可以直接左移就可以更上面的 bit 都不用處? 因為他每一回合有限定這會合是哪些 processor 在工作。也就是 participate 變數的工作數量。第一回合有兩個 processor 工作，也就是 rank 0 1。每一回合多一倍的工作數量。也因此更上面的 processor 都不用考慮，所以 exclusive or bitmask 才不用考慮高位元的 bit 是否相等以讓他們為 0。

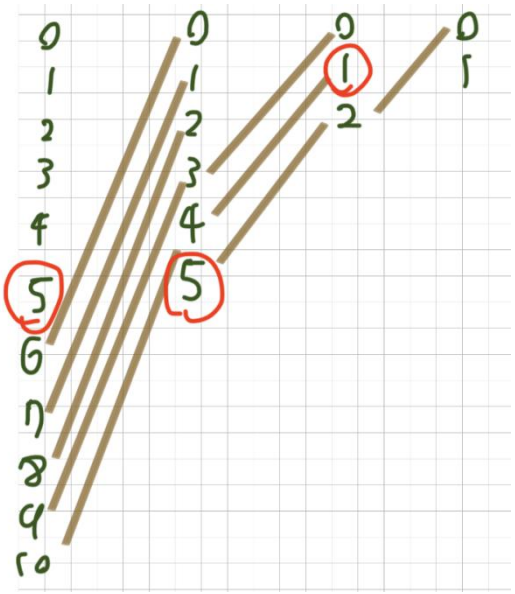
以上都還沒有提到不是 2 的冪次方的解決方法。



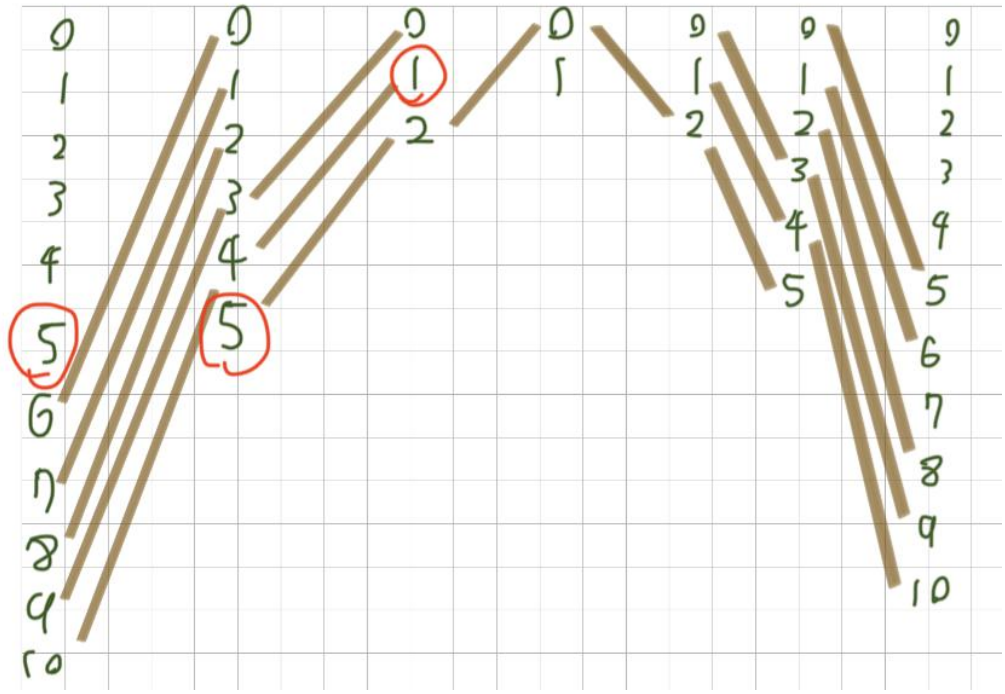
不是 2 的冪次是會多一回合處理大於 2 的冪次的 processor。向上圖會多出第四回合，處理 1000 1001 1010 的節點，而他們的 bitmask 為 1000 這樣 exclusive or (1000)會是 (0000、0001、0010)這些為傳送給他們的 processor。

這裡的樹狀結構跟前面將資料分散給 processor 的樹狀結構不一樣。這裡的迴圈是跟 2 的冪次做比較。像上圖 有 11 個 processor，所以會 while(bitmask < numprocs) 當 bitmask 為 1000 時仍會做 ，但當下一回合 10000 就大於總處理個數不會再繼續做 while 迴圈。

這種方法是先把多出來的部分都傳好，而之前的 processor 是留中間的節點等下一回合再傳。(如下圖)



為什麼不用分散計算的樹狀結構再將結果傳回去給每個 processor?



因為傳回去的時候不知道下一回合的 processor 數量為奇數或偶數，所以無法判斷是否 half + rm 的 rm 為 0 或 1，也就是原先的圈起來的紅色節點可能會沒被考慮到，而被認為是該回合要處理的 processor。

問題 2 : Analysis on your result

我比較(1)普通的執行方法跟(2)平行用 broadcast 的傳遞的和與(3)平行用樹狀結構傳遞的三種方法。

(1)普通的執行方法

```
F74109016@sivslab-pn1:~$ gcc problem2_sequential.c -o ./test
F74109016@sivslab-pn1:~$ ./test
Process 0 finished in time 29.177966 secs.

estimate pi is 3.141590.
```

(2)平行用 broadcast 的傳遞

```
F74109016@sivslab-pn1:~$ mpicc problem2_parallel.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 5 ./test
Process 0 finished in time 8.223388 secs.

estimate pi is 3.141682.

F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 17 ./test
Process 0 finished in time 7.175404 secs.

estimate pi is 3.141990.

F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 40 ./test
Process 0 finished in time 4.096077 secs.

estimate pi is 3.142440.
```

(3)平行用樹狀傳遞 (輸入是 10^9 相當於 2^30 次方)

```
F74109016@sivslab-pn1:~$ mpicc problem2.c -o ./test
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 5 ./test
Enter the number of intervals: (0 quits) 1000000000

estimate pi is 3.141562.

Process 0 finished in time 9.543324 secs.
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 17 ./test
Enter the number of intervals: (0 quits) 1000000000

estimate pi is 3.141568.

Process 0 finished in time 5.869556 secs.
F74109016@sivslab-pn1:~$ mpiexec -f hosts -n 40 ./test
Enter the number of intervals: (0 quits) 1000000000

estimate pi is 3.141522.

Process 0 finished in time 3.888947 secs.
```

分析
普通的執行方法
用時
29.177966

平行用 broadcast 的傳遞
用時
# of p = 5 : 8.223388
# of p = 17 : 7.175404
# of p = 40 : 4.096077(第二名)

平行用樹狀傳遞
用時
0f p = 5: 9.543324
of p = 17 : 5.869556 (第三名)
# of p = 40 : 3.888947 (第一名)

觀察一：
第二題要處理的資料量很大，當 processor 的數量越多，用時就越來越少。
無論是平行 broadcast 的傳遞，又或是平行用樹狀的傳遞都是。

觀察二：
因為這題處理的資料量很大，普通運算是用循序的時間在做計算 所以會比 平行慢很多。

觀察三：
樹狀結構平行 在 processor 數量=5 的時候比 broadcast 慢一點，理論上應該娉行樹狀傳遞都會比較快。我猜測可能是因為 processor 數量很少，有沒有分樹狀結構平行傳遞影響不大，反而是分很多層的方式做傳遞可能會浪費很多時間。

觀察四：
樹狀結構平行 在 processor 數量多的時候比 broadcast 快，可能因為 broadcast myid == 0 的 processor 要傳遞給很多 processor 造成時間的浪費。

問題 3 : Any difficulites?
這題在寫 Process 0 should read in the total number of tosses and broadcast it to the other processes by using point-to-point communications.時候比較麻煩，因為原本一直想用原本的樹狀結構來逆推回去，但發現怎麼改都很多 bug。後來上網找到這種做法，才回去反省原先的逆推回去的做法為什麼不可行，就了解原先的只能預估下一次處理的 processor 數量>=2*當前的 processor 數量，有沒有多一個會差很多。

這題在寫資料 input 的時候花了很多時間理解，最後寫成這樣。while 迴圈來讀值，同時將 MPI_Finalize 寫在 while 迴圈裡面。

可以說這題有 bitmask 的 broadcast 平行方式傳遞 number_of_tosses。的方式很神奇想了很久才想通，理解這個演算法。
而使用者 input 的時候原先都會 hang 住，也因此我改了很多種方法最後才成功。真的是一關難如一關。

問題 4 : (optional) Feedback to TAs

用 bitmask 將預測的 pi 值傳給每個 processor 的方式十分有趣，我很驚嘆竟然有人可以想出這麼好玩的演算法。