Hw2\_1

Processes數量與Data大小的差異與分析

|  |  |  |  |
| --- | --- | --- | --- |
| Process\Smooth | 10^3 | 10^4 | 10^5 |
| 2 | 25.351 | 556.114 | X |
| 4 | 12.932 | 373.614 | X |
| 8 | 6.938 | 121.086 | 692.89 |
| 16 | 4.473 | 50.32 | 376.727 |

(Smooth 10^4 次的時候似乎很多人同時連線，試了一整晚><)

Smooth次數

Process數量

時間(s)

從Smooth的10^3和10^5次來看，processes從8個變到16個分別快了1.5倍、1.8倍，發現Smooth次數越多，speed up的程度越明顯，可以推測Processes間的溝通、平行處理，佔用了一定的時間，當Data越來越大的時候，平行處理所額外消耗的時間的比重會逐漸降低

(10^4每次都不同，不參考

程式架構

因為第一列需要與最後一列的資料作平均，最後一列也需要與第一列作平均，因此將BMPSaveData多開兩列，並把readBMP中BMPSaveData的起始位置改成第一列，將圖檔第一列放到最後、圖檔最後一列放到最前面，方便一整塊scatterv給各個Processes。

在scatterv之前用Build\_mpi\_type，將RGBTRIPLE的三個unsigned char(rgbBlue, rgbGreen, rgbRed)組成一個MPI中可用的type，方便scatterv。

Process 0 scatterv 給各個processes他們需要處理的biHeight/process number列，並多給前一列及後一列，以讓需要處理的第一列及最後列有資料可以取用平均。

每個Processes處理完一次Smooth之後就將處理過的第一列及最後列與id-1、id+1的Process做資料交換更新

最後smooth次數全部做完之後，將處理完的資料gatherv回Process 0(提供資料以平均的兩列就不回傳)

Hw2\_2

程式架構

randNum 用來儲存產生的亂數

RecvNum 用來接收來自別的Processes的資料

tmpNum 用來暫存process間sort的結果

process 0 讀入總共需要排序的數後，broadcast給各個processes，各自在自己的randNum陣列中產生需要的亂數，而為了方便統一長度，在較少亂數的processes中，補上RAND\_MAX以讓各processes的randNum中的數字一樣多

各個processes先各自做odd-evens sort，gatherv到process 0先印出local list

再來做processes之間的odd-even sort，當process 0 與 process 1互為partner時，我將彼此的randNum整列傳給對方。

process 0即從自己的randNum和收到的RecvNum的index 0 開始比較，將較小的數字一個一個存入tmpNum。

process 1 即從自己的randNum 和 RecvNum 的 index (長度-1) 開始比較，將較大的數字從tmpNum 的index (長度-1) 開始往回存入。

得到新數列後將tmpNum與randNum互換，randNum現在即為最新的值

最後processes間Sort完後傳回Process 0，印出剛好數量的數列。

(後面剩下的都是一開始多補的RAND\_MAX)