HW2

資訊 112 吳信葆

第一題

程式

pseudo code

```
Proc 0  讀  input.bmp
Proc 0  廣播 bmpInfo.Height(all_count)  給其他  procs，算出  local_count
Proc 0  用  Scatterv  分配  bmp  檔給其他  proc
Procs  平滑運算  n  次
    Local  計算
    交換邊界(非迴圈)
        奇數核心將上界給偶數核心
        偶數核心將上界給奇數核心
        奇數核心將下界給偶數核心
        偶數核心將下界給奇數核心

Procs Gatherv  到  proc 0
儲存檔案
```

交換邊界

這邊分奇偶是因為如果大家一開始先一起 send 的話沒人能 recv ，因為大家都在等自己的下一個去 recv。

結果分析

指令

mpiicpc h2_problem1.cpp -o h2_problem1.out

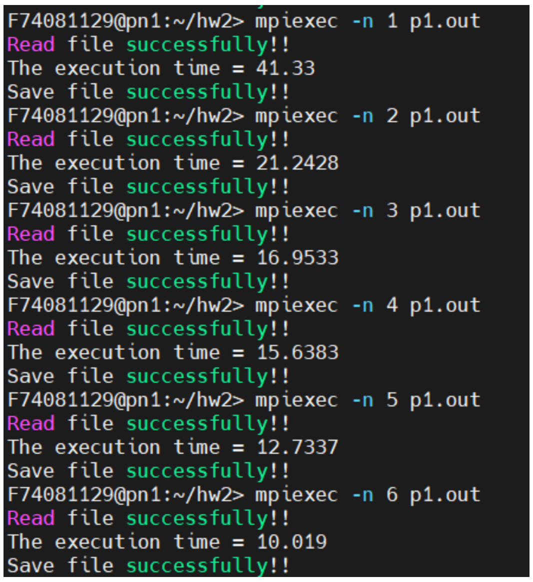
mpiexec -n 8 h2_problem1.out 1000

解釋

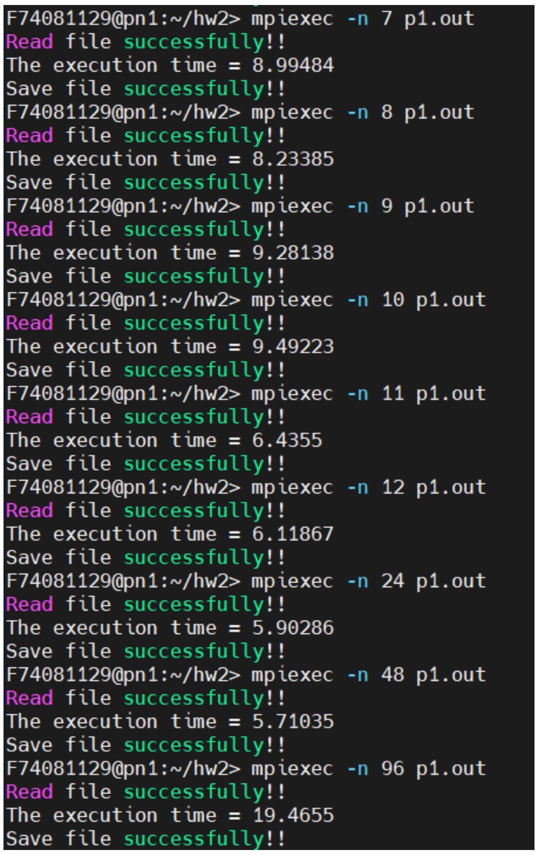
第二條後面有 1000 代表會作一千次平滑畫(沒打就默認是 1000)。

輸入檔 input.bmp ，輸出檔 output.bmp，寫死在程式裡。

結果(更改核心數)



```
F74081129@pn1:~/hw2> mpiexec -n 1 p1.out
Read file successfully!!
The execution time = 41.33
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 2 p1.out
Read file successfully!!
The execution time = 21.2428
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 3 p1.out
Read file successfully!!
The execution time = 16.9533
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 4 p1.out
Read file successfully!!
The execution time = 15.6383
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 5 p1.out
Read file successfully!!
The execution time = 12.7337
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 6 p1.out
Read file successfully!!
The execution time = 10.019
Save file successfully!!
```



```
F74081129@pn1:~/hw2> mpiexec -n 7 p1.out
Read file successfully!!
The execution time = 8.99484
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 8 p1.out
Read file successfully!!
The execution time = 8.23385
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 9 p1.out
Read file successfully!!
The execution time = 9.28138
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 10 p1.out
Read file successfully!!
The execution time = 9.49223
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 11 p1.out
Read file successfully!!
The execution time = 6.4355
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out
Read file successfully!!
The execution time = 6.11867
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 24 p1.out
Read file successfully!!
The execution time = 5.90286
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 48 p1.out
Read file successfully!!
The execution time = 5.71035
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 96 p1.out
Read file successfully!!
The execution time = 19.4655
Save file successfully!!
```

可以看到大部分的測試符合核心越多越快，12 核以上就看不出明顯趨勢。

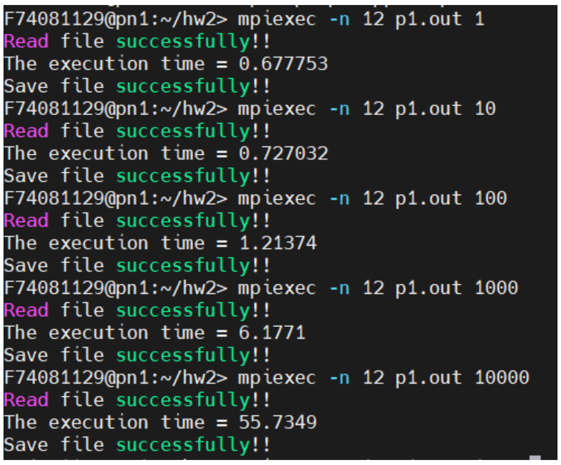
12 核以上太多的話時間會有點不穩，可能是單核要頻繁換 task 造成。

將 核心數 * 時間 得到的直應該要差不多，相當於單核完成時間:

核心數	1	2	4	12
核心數 * 時間	41.334	42.4856	62.5532	73.42404

沒有一樣代表說 cpu 之間溝通也需要時間，而可以看到值不斷上升，則可看出這個時間會跟 cpu 間呈現正相關。

結果(更改 NSmooth 平滑處理次數)



```
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out 1
Read file successfully!!
The execution time = 0.677753
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out 10
Read file successfully!!
The execution time = 0.727032
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out 100
Read file successfully!!
The execution time = 1.21374
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out 1000
Read file successfully!!
The execution time = 6.1771
Save file successfully!!
F74081129@pn1:~/hw2> mpiexec -n 12 p1.out 10000
Read file successfully!!
The execution time = 55.7349
Save file successfully!!
```

可以看到時間遞增，畢竟執行次數越來越多，接著我們將 執行時間 /NSmooth，類似於平均一次平滑化的時間。

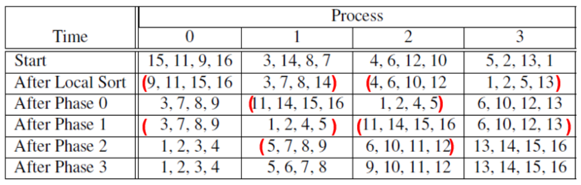
NSmooth(n)	1	10	100	1000	10000
Time/n	0.677753	0.0727032	0.0121374	0.0061771	0.00557349

可以看到越來越少，因為我們知道時間還包刮 Gatherv 和 Scatterv，而這個時間可能跟核心數比較有關，跟次數較無關。

第二題

程式

照著 ch3 p140,141，實作



Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	(9, 11, 15, 16	3, 7, 8, 14)	(4, 6, 10, 12	1, 2, 5, 13)
After Phase 0	3, 7, 8, 9	(11, 14, 15, 16	1, 2, 4, 5)	6, 10, 12, 13
After Phase 1	(3, 7, 8, 9	1, 2, 4, 5)	(11, 14, 15, 16	6, 10, 12, 13)
After Phase 2	1, 2, 3, 4	(5, 7, 8, 9	6, 10, 11, 12)	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

Local sort 是 O(nlogn)的 c++ sort (#include<algorithm>)

Merge 是直接將自己的陣列傳給 partner，讓他們自行 merge O(n)

T(n) 時間複雜度是 (n/k)log(n/k)(local sort) + (n/k)(local merge)*k(number of phase)，當 n 無窮且每次傳送是常數複雜度時，T(n)=O(nlogn/k)。

結果分析

測試

command:　mpiicpc h2_problem2.cpp -o h2_problem2.out

command:　mpiexec -n 4 h2_problem2.out

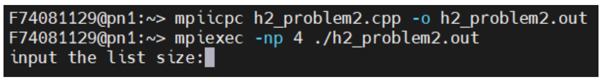
Stdin: 1000

第一行編譯

第二行執行

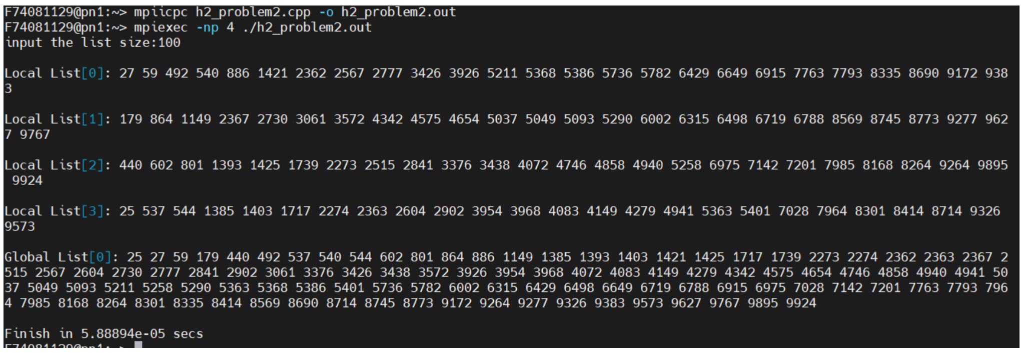
第三行是一個數字，也就是程式(proc0)一開始會用 cin>> 讀一個數字，該數字為 global list 的大小

結果(n=100)



```
F74081129@pn1:~> mpiicpc h2_problem2.cpp -o h2_problem2.out
F74081129@pn1:~> mpiexec -np 4 ./h2_problem2.out
input the list size:
```

輸入一個數字(100)

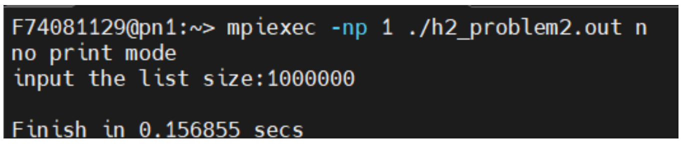


```
F74081129@pn1:~> mpiicpc h2_problem2.cpp -o h2_problem2.out
F74081129@pn1:~> mpiexec -np 4 ./h2_problem2.out
input the list size:100

Local List[0]: 27 59 492 540 886 1421 2362 2567 2777 3426 3926 5211 5368 5386 5736 5782 6429 6649 6915 7763 7793 8335 8690 9172 938
3

Local List[1]: 179 864 1149 2367 2730 3061 3572 4342 4575 4654 5037 5049 5093 5290 6002 6315 6498 6719 6788 8569 8745 8773 9277 962
7 9767

Local List[2]: 440 602 801 1393 1425 1739 2273 2515 2841 3376 3438 4072 4746 4858 4940 5258 6975 7142 7201 7985 8168 8264 9264 9895
 9924

Local List[3]: 25 537 544 1385 1403 1717 2274 2363 2604 2902 3954 3968 4083 4149 4279 4941 5363 5401 7028 7964 8301 8414 8714 9326
9573

Global List[0]: 25 27 59 179 440 492 537 540 544 602 801 864 886 1149 1385 1393 1403 1421 1425 1717 1739 2273 2274 2362 2363 2367 2
515 2567 2604 2730 2777 2841 2902 3061 3376 3426 3438 3572 3926 3954 3968 4072 4083 4149 4279 4342 4575 4654 4746 4858 4940 4941 50
37 5049 5093 5211 5258 5290 5363 5368 5386 5401 5736 5782 6002 6315 6429 6498 6649 6719 6788 6915 6975 7028 7142 7201 7763 7793 796
4 7985 8168 8264 8301 8335 8414 8569 8690 8714 8745 8773 9172 9264 9277 9326 9383 9573 9627 9767 9895 9924

Finish in 5.88894e-05 secs
```

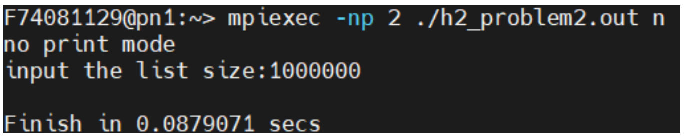
該程式測時不包含印出時間和變數宣告，但包含預處理。

結果(n=100000)

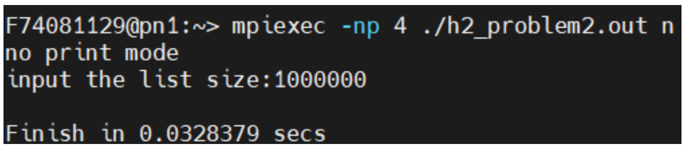
我在 list size= 十萬做測試，我的程式有一個參數 “n”，輸入會開啟 no print mode，不會印出 list 方便做效能測試。



```
F74081129@pn1:~> mpiexec -np 1 ./h2_problem2.out n
no print mode
input the list size:1000000

Finish in 0.156855 secs
```



```
F74081129@pn1:~> mpiexec -np 2 ./h2_problem2.out n
no print mode
input the list size:1000000

Finish in 0.0879071 secs
```



```
F74081129@pn1:~> mpiexec -np 4 ./h2_problem2.out n
no print mode
input the list size:1000000

Finish in 0.0328379 secs
```

由於 pn2 pn4 (比較慢，會拖後腿)不穩定，我們都是以最多 4 顆核心做測試，可以發現時間真的越來越少，我們平行化相當成功。