HW5 report

F74081129 資訊 112 吳信葆

h5_problem1

問題回答

1. If we try to parallelize the for i loop (the outer loop), which variables should be private and which should be shared?
   Ans.

```
#pragma omp parallel for num_threads(np) default(none)
private(count) \
    shared(len) shared(a) shared(temp)
    for (int i = 0; i < len; i++)
    {
        count = 0;
        for (int j = 0; j < len; j++)
            if (a[j] < a[i])
                count++;
            else if (a[j] == a[i] && j < i)
                count++;
        temp[count] = a[i];
    }
```

   記本上我們要分別算舊元素的新位置在哪，所以除了 count 設 private，其他設 shared 就好。
2. If we parallelize the for i loop using the scoping you specified in the previous part,are there any loop-carried dependences? Explain your answer.
   基本上，該演算法只依賴舊陣列(a)的值，新陣列(temp)元素彼此互相獨立，也就是 for I loop 是可以被平行化的，因為 for I loop 沒有 loop dependence。
3. Can we parallelize the call to memcpy? Can we modify the code so that this part of the function will be parallelizable?
   當我們把 code 改完後就可以，先算出每個 thread memcpy 的起始位置，再算出 memcpy count 即可。

```
    disp = (int*)malloc(np * sizeof(int));
    cnts = (int*)malloc(np * sizeof(int));
    disp[0] = 0;
    cnts[0] = len / np + (id < (len % np));
    for (int i = 1; i < np; i++)
    {
        cnts[i] = len / np + (id < (len % np));
        disp[i] = disp[i - 1] + cnts[i - 1];
    }
#pragma omp parallel num_threads(np) default(none) \
    shared(a) shared(temp) shared(len) private(id)
shared(cnts) shared(disp)
    {
        id = omp_get_thread_num();
        memcpy(a + disp[id], temp + disp[id], cnts[id] *
sizeof(int));
    }
```

4. Write a C program that includes a parallel implementation of Count sort.
   In ~/hw5/p1/h5_problem1.c
5. How does the performance of your parallelization of Count sort compare to serial Count sort? How does it compare to the serial qsort library function?
   速度: serial count sort < Parallelization count sort < serial qsort

測試及分析

環境是在我的筆電 wsl，4 核 8 線程。
老師伺服器的資料夾: ~/hw5/p1/

```
gcc h5_problem1.c -o h5_problem1.out -fopenmp -std=c99
./h5_problem1.out
```

可以使用 Makefile

排序陣列大小 l= 10000
單核(serial count sort)(n=1)

```
./h5_problem1.out -n 1 -l 10000
4 984 984 984 984 984 984 984 984
989 989 989 989 989 989 989 989 9
3 993 993 994 994 994 994 994 994
996 996 997 997 997 997 997 997 9
The execution time = 0.445109
```

多核平行(Parallelization count sort)(n=4)

```
./h5_problem1.out -n 1 -l 10000
```

```
4 984 984 984 984 984 984 984 9
989 989 989 989 989 989 989 989
3 993 993 994 994 994 994 994 9
996 996 997 997 997 997 997 997
The execution time = 0.142947
```

你也可以直接 ./h5_problem1.out，這是默認 case(n=4,l=10000)。

QSort(serial qsort)(n=1)(-q 代表 qsort, 無視 -n 參數)

```
./h5_problem1.out -q -l 10000
```

```
989 989 989 989 989 989 989 989
3 993 993 994 994 994 994 994 9
996 996 997 997 997 997 997 997
The execution time = 0.000856
```

這個結果告訴我們，在做平行處理前需要先做**時間複雜度**分析分析。

假定:

資料大小: n

#thread: t

時間複雜度:

Serial count sort: O(n*n)

Parallelization count sort(t=4): O(n*n/4)=O(n*n)

serial qsort: O(nlgn)

無論多少電腦，(速度)**O(nlgn)>>O(n*n)**，但是如果要平行化又要有 O(nlgn) 的時間，hw2 先做 c++ local sort(nlgn) 再做 odd even merge(n*t) 是不錯的選擇。

Time (nlgn)/t + nt

n=1000000,t=4

```
paul@LAPTOP-4VQ86L63:/mnt/c/Users/Paul/OneDrive/桌面/平行/hw2$ mpiexec -np 4 ./h2_problem2.out
input the list size:1000000
```

```
Finish in 0.0633979 secs
```

QSort

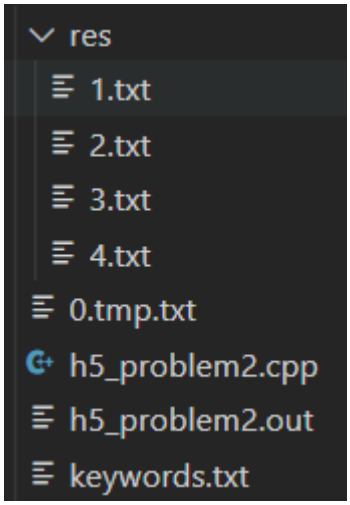
```
./h5_problem1.out -q -l 1000000
```

```
The execution time = 0.103738
```

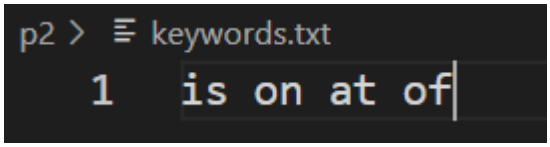
h5_problem2

What have you done

題目要求都有完成，我還做了當要讀取的檔案數量遠高於 #producer，我會以 for 迴圈循序分配要讀取的檔案，沒設定 #producer 則會跟檔案數量相同。



默認架構長這樣，0.tmp.txt 是 res/裡的 txt 檔，程式自動產生。
res 是 producer 要讀的 collection of files 只讀第一層，不遞迴，keywords.txt 存關鍵字，空格隔開。



Analysis on your result

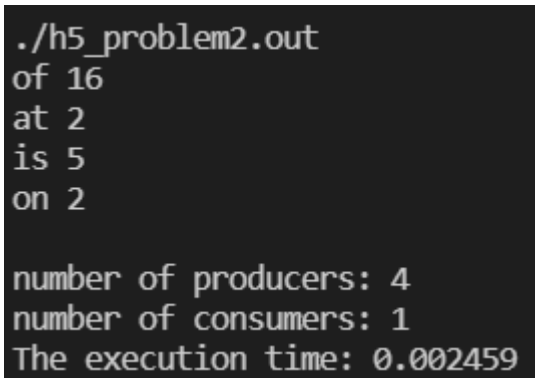
環境同第一題
老師伺服器資料夾: ~/hw5/p2/
編譯: g++ h5_problem2.cpp -o h5_problem2.out -fopenmp -std=c++0x
執行: ./h5_problem2.out
可以使用 Makefile

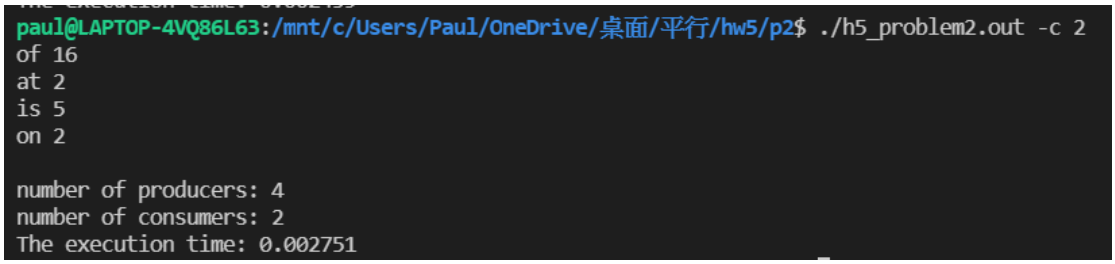
參數	定義	默認
-p	Producer 數量	檔案總數
-c	Consumer 數量	4
-k	Keywords 檔案	./keywords.txt
-f	Collection of files 資料夾	./res

範例: ./h5_problem2.out -f res/ -c 4 -p 4 -k keywords.txt

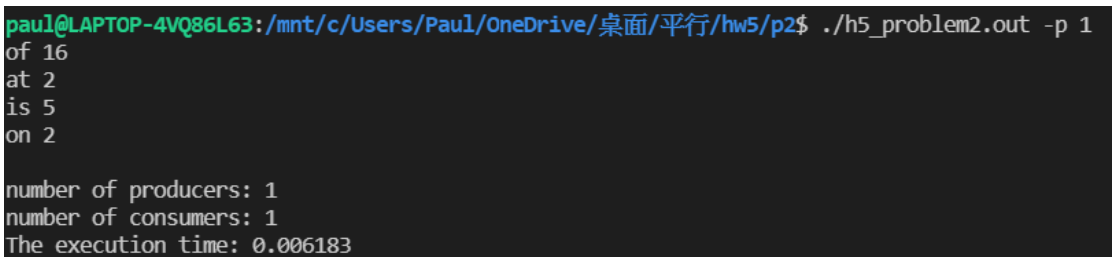
默認的 consumer 我因為要做測試，不能用太多 thread，所以 Consumer 數量默認是 1，但為了讓助教一開始輸入命令就是平行化的，在老師伺服器改成 4，另外我的程式最後會顯示 producer consumer 數量，助教也可以用(-c -p)去調整數量。

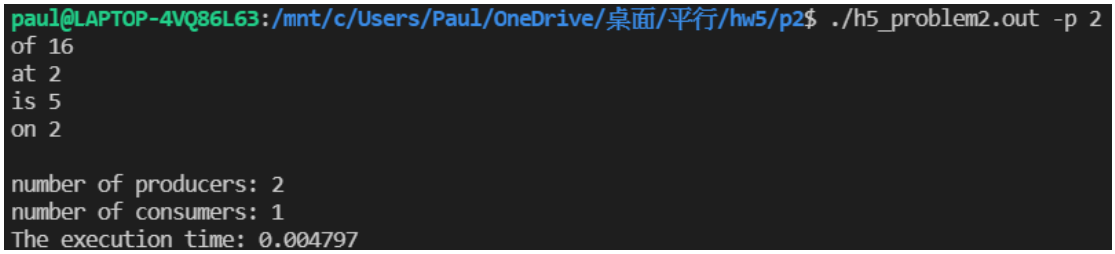
接著做速度測試





可以看到結果完全沒變，試著修改 #producer (p=1,p=2)





#producer 對結果還是有影響的，應該是供不應求，consumer 都在 idle 等資料。
我猜測該程式瓶頸可能是在讀檔，於是我把 consumer code 註解，包括從 queue 中取資料。
(對於我的 code, 相當於把 consumer 數量設 0)

```
paul@LAPTOP-4VQ86L63:/mnt/c/Users/Paul/OneDrive/桌面/平行/hw5/p2$ ./h5_problem2.out -c 0
is 0
on 0
at 0
of 0

number of producers: 4
number of consumers: 0
The execution time: 0.003651
```

可以看到 consumer 有做沒做時間差不多，證明我們的推論是對的。
結果圖表

(#p,#c)	(4,1)	(4,2)	(1,1)	(2,1)	(4,0)
time	0.002459	0.002751	0.006183	0.004797	0.003651

Any difficulties?

當我們存取 shared queue ，需要進 critical session。

假設一種情形，我們檢測該 queue 有資料，進到 critical session 把資料 pop 出來，就有可能(大概率)出錯，因為可能有多個 consumer “同時” 檢測到該 queue 不為空(ex. Size=1,但有兩個 consumer)，而分別進到 critical session，然而根本沒有那麼多資料可以 pop(第一個 pop 成功，第二個 segmentation fault)。

解法很簡單，就是在 critical session 裡再檢測一次，這個 code 我寫在 pop()。