Programming Assignment #4 Report

B03902048 林義聖

實驗數據

- Machine: CSIE Workstation linux3
- CPU: 24 cores
- Memory: 128 GB

In the first two experiments (n = 100, 10000), I let the output display on the screen. In the last two experiments (n = 1000000, 10000000), I redirect the output, so it speed up a lot.

- n = 100

segment size	n / 100 1	n / 25 4	n / 10 10	n / 5 20	n / 2 50	n / 1 100
real	0.056	0.020	0.010	0.008	0.004	0.003
user	0.000	0.000	0.000	0.000	0.000	0.000
sys	0.020	0.008	0.004	0.004	0.000	0.000

- n = 10000

segment size	n / 100 100	n / 25 400	n / 10 1000	n / 5 2000	n / 2 5000	n / 1 10000
real	0.058	0.038	0.036	0.037	0.022	0.020
user	0.045	0.028	0.028	0.022	0.015	0.016
sys	0.016	0.006	0.004	0.008	0.008	0.000

- n = 1000000

segment size	n / 100 10000	n / 25 40000	n / 10 100000	n / 5 200000	n / 2 500000	n / 1 1000000
real	5.098	3.737	3.419	2.705	2.261	1.912
user	3.173	2.748	2.417	2.049	1.645	1.272
sys	0.198	0.164	0.132	0.080	0.076	0.045

- n = 10000000

segment size	n / 100 100000	n / 25 400000	n / 10 1000000	n / 5 2000000	n / 2 5000000	n / 1 10000000
real	44.947	36.108	28.918	25.474	25.700	19.897

segment size	n / 100 100000	n / 25 400000	n / 10 1000000	n / 5 2000000	n / 2 5000000	n / 1 10000000
user	30.988	28.935	23.969	20.721	17.94	14.154
sys	1.889	1.871	1.309	0.976	0.790	0.497

我對於上面的結果不太滿意，之後，我重新做了實驗。這次，由於我已經確認輸出結果是完全正確的，於是我將 output 全部拿掉，並且寫了 python script 對於 n = 1000000 和 n = 10000000 的兩項測驗的每一個 segment size 都跑了 20 次並取執行時間平均以求得準確的結果。

- n = 1000000

segment size	n / 100 10000	n / 25 40000	n / 10 100000	n / 5 200000	n / 2 500000	n / 1 1000000
real	0.550	0.500	0.520	0.560	0.670	0.910
user	1.300	1.240	1.170	1.060	0.950	0.890
sys	0.080	0.050	0.040	0.030	0.020	0.010

- n = 10000000

segment size	n / 100 100000	n / 25 400000	n / 10 1000000	n / 5 2000000	n / 2 5000000	n / 1 10000000
real	5.090	5.120	5.340	5.960	7.250	9.720
user	14.510	13.970	13.370	11.950	10.680	9.530
sys	0.670	0.560	0.460	0.370	0.300	0.190

結果觀察

前面的實驗結果因著 output 太多的關係，並沒有一個好的結果看出 multi-thread 的優勢。而由後面兩組 (12 個組合) 的實驗結果，可以明顯看出：thread 開得越多，則 user time 花得越多，而 real time 大致上為 thread 越多則越快。thread 開得多時，每一個 thread 的執行時間都算在 user CPU time 裡面。所以，雖然 multi-thread 似乎讓 real time 減少了，實則，花了更多 CPU 時間，亦即花了共多的系統資源，在解決同樣的問題上。另一方面，我其實知道，自己寫的 merge sort 的效率是比不上 std::sort()，但是在 multi-thread 的情況下，則可以贏過 std::sort()，代價除了較多的 user CPU time，也包括開 multi-thread 造成的 overhead，使 sys CPU time 同樣增加了不少。不過就最後的結果而論，multi-thread 是有明顯助益的。segment size 為 n 或是 n / 100，由最後的結果可以發現，real running time 差了近兩倍，顯然有達到 multi-thread 應有的果效了。

附上測試用的腳本：

```
#!/usr/bin/python3

import os
import subprocess

thread_amount = [100, 25, 10, 5, 2, 1]
filename = ["big_data", "large_data"]
inputsize = [1000000, 10000000]

for i in range(2):
      print("")
      print("filename = " + filename[i] + ", input size = " + str(inputsize[i]))
      print("-------------------------------------")

      for j in range(6):
            segment_size = inputsize[i] / thread_amount[j]
            print ("segment_size = %d" % (segment_size) )
            cmd = "/usr/bin/time -f \"%e,%U,%S\" ./merger "+
str(int(segment_size)) +" < "+ filename[i] +" > /dev/null"

            real_time = 0.0
            user_time = 0.0
            sys_time = 0.0

            times = 20
            for k in range(times):
                  process = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
                  output, error = process.communicate()
                  line = error.decode("utf-8")
                  result = line.split(',')
                  real_time += float(result[0])
                  user_time += float(result[1])
                  sys_time += float(result[2])

            real_time /= float(times)
            user_time /= float(times)
            sys_time /= float(times)
            print("real %.3f" % round(real_time,2) )
            print("user %.3f" % round(user_time,2) )
            print("sys  %.3f" % round(sys_time,2) )
```