HW1 Document 資訊三乙 10727219 許志仲

1. 開發環境

Windows 10

Python 3.7

import time, threading, multiprocessing

另外使用numba的jit()進行加速

1. 實作方法和流程
2. 將指定文件之資料全部讀入並存入List，直接進行BubbleSort。
3. 複製先前讀入的List，並平均切成K份，在for迴圈內新增thread對個別list進行BubbleSort，之後持續檢查，只要List大於一份，就持續迴圈呼叫新增thread將資料兩兩進行Merge。
4. 複製先前讀入的List，並平均切成K份，呼叫一個大小為K的multi-process Pool對list中的每一份進行BubbleSort，之後持續檢查，只要List大於一份，就持續迴圈呼叫一個大小為K/2的multi-process Pool將list的資料兩兩進行Merge。
5. 主程式本體所在可視為一Process，因此直接在main()之下執行，先複製先前讀入的List，並平均切成K份，在for迴圈內個別list進行BubbleSort，之後持續檢查，只要List大於一份，就持續迴圈呼叫將資料兩兩進行Merge。
6. 四種做法進行比較

因方法1的數據較其他方法差異較大，將折線圖分開表示，原則上當資料量增加到一定程度時，方法1所需的時間會遠遠大於其他三種，而其他三種方法速度相對快許多，在資料量為100萬時，方法4>2>3。

速度(快->慢) 4>2>3>1

時間隨資料量成長幅度(大->小) 1>2>4>3

1. 分析結果和原因

方法1的結果算是預料之內，畢竟bubblesort的缺點就是資料量一大就會變慢許多，但是方法2 3 4的結果就比較奇怪，方法3雖然較久但是成長幅度較2 4小，理論上multi-process應該最快，研究後發現如果拿掉為了省時兒加上的numba @jit指令，multi-process就會是最快的。最後方法2與4比起來大致相同。