作業二 threaded sorting

404410030 資工三 鄭光宇

編譯：

make

（編譯完會先做自我測試，與沒有平行化的版本輸出結果比較）

執行：

./hw2 rand_seed array_size

rand_seed: 隨機數種子

array_size: 欲排序陣列大小

1. 如何利用平行化提升速度？

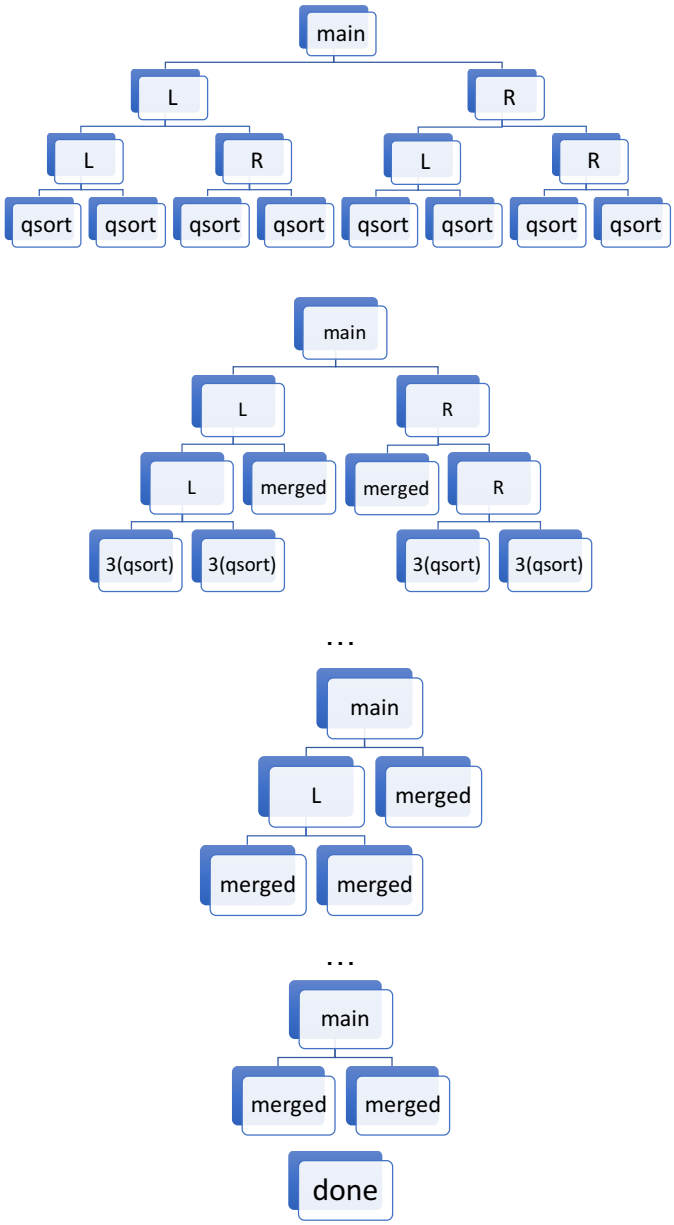
   因為是使用 merge sort，在執行分割時，會有很多獨立的區間，不會互相重疊，也沒有先後的相依性，所以可以同時執行，經由平行化提升速度。

2. 如何確保不會發生 race condition？

   因為在分割時，陣列中每個元素只會被一個 thread 讀寫，並且在 merge 前，會等待底下分割出來的所有 thread 都做完，再執行 merge，所以不會發生 race condition。

3. 使用圖形說明你的程式碼是很有效率的，簡述使用的演算法。

   使用一般的 merge sort，原先分割的遞迴呼叫改成建立一個新的 thread，等到遞迴到了指定深度後，改用內建 qsort。
   深度通常設定為 log2(核心數)，這樣設定可以預估最多同時使用 CPU 的 thread 數量，例如 8 核心的最大深度就是 3（從 0 開始算）。
   簡單的流程：
   甲、如果陣列大小小於 2，直接結束。
   乙、如果達到指定深度，改使用 qsort 排序，結束。
   丙、Parent thread 分割陣列 L, R
   丁、建立新的 child thread <- L，執行陣列左邊的 merge sort
   戊、建立新的 child thread <- R，執行陣列右邊的 merge sort
   己、主要 thread 等待兩個 child thread 做完
   庚、主要 thread 執行 merge
   辛、結束。



一些實作細節：

因為 merge sort 在 merge 時需要額外 O(n) 的空間，所以與其在 merge 時，才分配記憶體、merge、複製回原來 array，不如直接分配一塊跟整個 array 一樣大小的記憶體，然後在 merge 時直接拿來使用，就可以降低呼叫 malloc() 等 function 的 overhead，因為有大量的資料需要搬移，所以儘量讓分配到的記憶體對齊，可以提升速度。

沒有使用 mutex 和 semaphore 限制 thread 數量，因為最大出現的 thread 數量可以預先知道。如果有 thread 做完工作，代表他的 parent parent 會把沒在使用的 CPU 撿起來用。

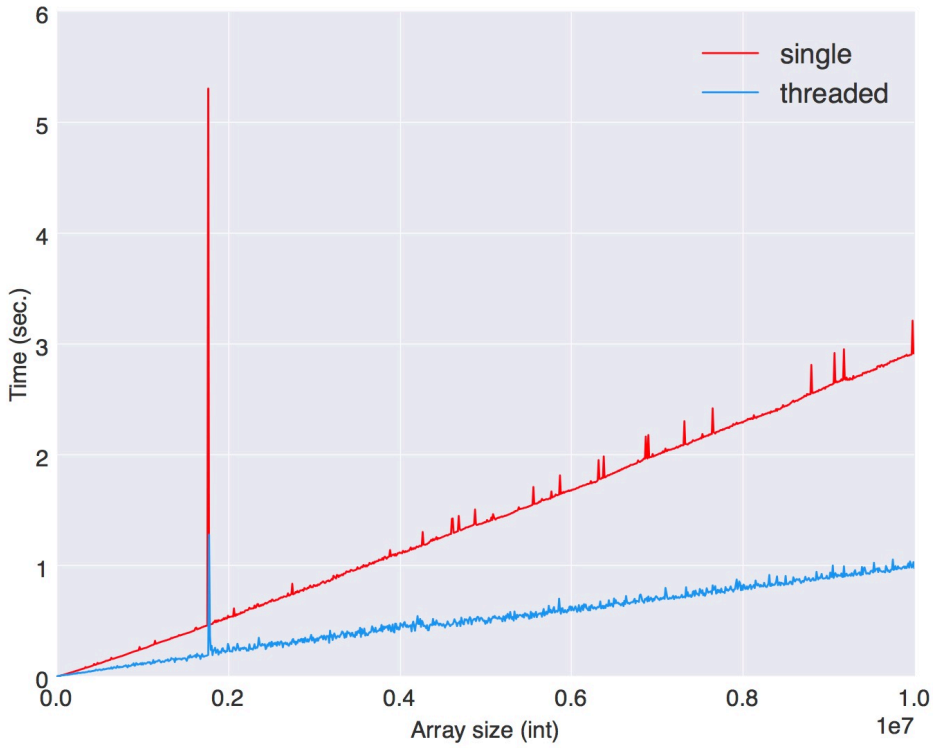
對於最後陣列前後元素相減、加總的部分，平行化的方式很單純，就是把排序好的陣列分成很多小部分，每個小部分都用 thread 同時去做前後元素相減、加總，最後在 main thread 的部分等待每個 thread 把答案傳回來，並加總，在小部分跟小部分間還有沒完成的，就順手在 main thread 完成，不會發生 race condition，因為每個 thread 負責的區塊不會有重疊的區間。速度快，因為各個 thread 間不會互相等待，同步的部分由 main thread 處理。

結果：

對於不同陣列大小，分別測試執行時間。

single 是直接呼叫 qsort()，不使用任何平行化機制。

threaded 是這支程式，在排序、加總上做平行化。



由上圖可以看出，相較單執行緒提升約 2~3 倍速度。

註記：

排序的 function 寫在 msort.*

加總的 function 寫在 threadedsum.*