2023 Parellel programming HW1

109062113 葛奕宣

Implementation

How do you handle an arbitrary number of input items and processes?

我考慮兩種情況，process 大於等於 n 或小於 n。

若大於等於 n，我就把數字平均分散在前 n 個 process。

若小於 n，我就把數字先除以總 process 數，再把餘數分配在前 k 個 process 上。

How do you sort in your program?

我用 boost::spreadsort，這是我在網路上查到的一個 library，我自己實際測試上比 std::sort 快非常多。

Detail:

每個 process 會有三個 float array，my_array, received_array, result_array，分別代表自己維護的 array、將要收到的來自左右 neighbor 的 array、用來存 merge sort 結果的 array。

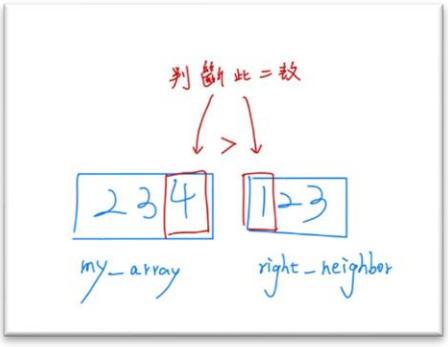
程式一開始會讀取自己的 my_array，用預先算好的 start 跟自己的 size，用 MPI_input 讀。

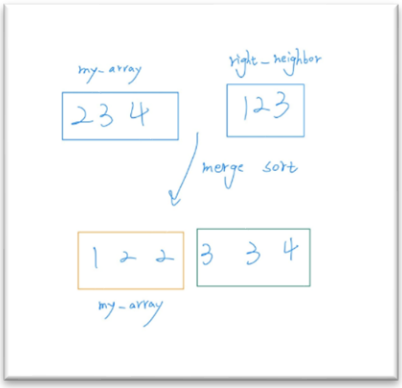
Start 的部分要注意若是 array_size>process 的情況，計算上要特別注意會有兩種 array_size。

接著會 sort my_array，用 spreadsort。

接著要計算迴圈以及溝通所需的參數。我定義在 odd-even sort 時，左右的 rank 是左右 neighbor，並且要分別計算他們的 array_size，我的變數叫做 left/right_count，之後用 MPI_Sendrecv 都必須 specify 這些大小，所以要計算。

而我的 odd-even sort 方法是，依照 odd, even 輪，先判斷需不需要跟 neighbor swap，判斷方法是去看兩個 array 中最左與最右(左圖)的數字是否符合遞增，這裡會需要第一次的 MPI_Sendrecv，如果需要 swap，則會用 MPI_Sendrecv 傳送自己的 array 並接受 neighbor 的 array，並在本地將這兩個 array merge sort 完，再取自己的一部分存在自己的 my_array(右圖)。



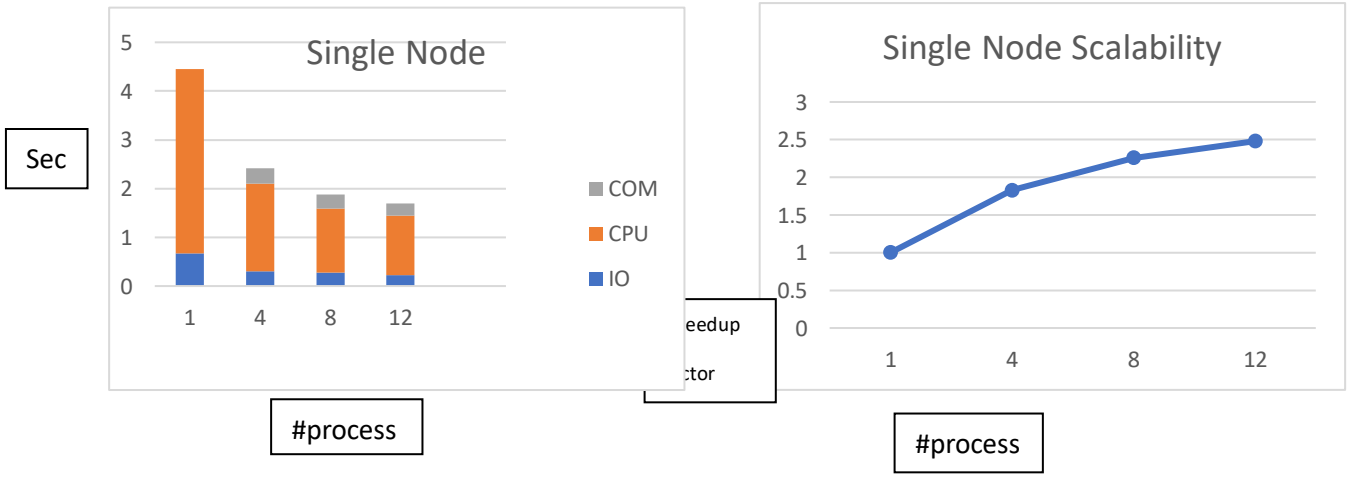


而判斷迴圈結束，我是去判斷每一個 process 有沒有進行 swap，我開了兩個變數 sort, all_sort，sort 初始化為 1，當 swap 發生，sort=0，而我會在迴圈最後用 all_reduce 把 sort 都加起來成 all_sort，所以 while 迴圈的判斷條件就是 all_sort<process 數，代表有 process 還在發生 swap。程式的最後會用一開始算的 start 跟 my_array_size 用 MPI_output 把 array 寫到對應的位置，最後再 delete array 記憶體並 Finalize。

Experiment & Analysis

Performance Metrics:
我在 code 裡面用 MPI_Wtime()分別計算 IO, communication time, IO 我包住讀寫 file 的部分，communication 我則是包住 sendrecv，我再用 ipm 去讀 wallclock time 得到總時間，再用總時間扣掉二者得到 CPU time.
IPM command:
IPM_REPORT=full IPM_REPORT_MEM=yes IPM_LOG=full LD_PRELOAD=/opt/ipm/lib/libipm.so
srun -N1 -n1 ./hw1 64123483 30.in 30.out

在 single-node 的情況下，我用 testcase 30 當作我的測資，因為他在單一 node，單一 process 的情況下跑了 4 秒，這個數字讓我在 scale 越大時，不會因為數字過小計算出現誤差。
在取得數字時，我會將每個 process 的時間加起來平均 ，並且整個測試會跑 2-5 次，確保沒有太大誤差的產生，因為我在實驗時就有出現過 IO 時間突然變很大等例外狀況。
我首先測試 single node 的 communication, IO, CPU time，可以看到這隻程式就是 bound by CPU，因為大部分的時間都花在 CPU time，而我猜測因為 single node 的原因，communication 的時間沒有很顯著，而從圖表可以看到，當 process 增加，cpu time 跟著減少，這就是因為程式被平行優化了，而 IO 也跟著減少，我猜測是因為每個 process IO 要讀的檔案大小減少了，所以不會被 IO 卡住，再來是 communication time，是隨著 process 增加而增加，這也合理是因為溝通的次數增加了，但這個增加的幅度小於 CPU 的優化程度，所以整體時間是下降的。
接下來觀察 Single Node Scalability，斜率隨著 process 增加而逐漸平緩，可以看出 speedup factor 的增加幅度趨緩，我認為在 single node 的情況下，我的程式 scalability 還行，因為就算從 8 process 到 max 12 process，還是有約 0.25 的上升。

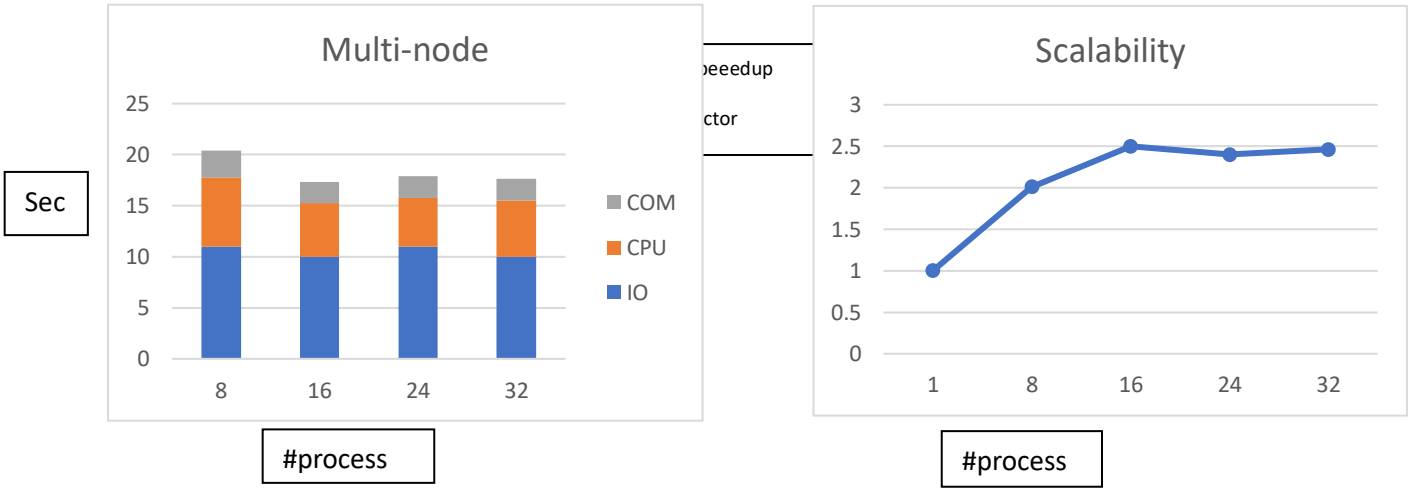


Multi-Node:

在測試 Multi-node 時，我選用 8 node，process 選擇 8~32，測資選擇 N 較大的 35，選擇 8node 是因為我希望在 process 很多的情況下每個 process 的資源都是充足的，CPU time 的優化會比較不受影響。

可以看到在 8 跟 16 process 時，CPU_Time 的減少時間還算顯著，而在 16 process 之後，整體的優化已經不明顯，甚至 IO, communication 都有增加的可能，而 CPU time 則是趨平。
這代表此程式在越高 process 數量時，優化不是一直向上的。
而 Scalabiltity 的圖更是說明如此狀況，在過了 16 process 後，speedup factor 還有下降的可能，所以整體來說此程式在 multi-node 的 scalability 不算優。

並且 IO 時間是大於 cpu 時間的，所以在 multi-process 的情況下，IO 有可能會成為 bottleneck。



Discussion:

Compare I/O, CPU, Network performance. Which is/are the bottleneck(s)? Why? How could it be improved?

根據 single-node 的圖，CPU time 佔的時間最長，所以 bottleneck 是 cpu time，要優化就是透過平行分散 array 讓每一個 process sort 的數量降低，然而要注意 IO，根據 multi-node 的圖，在 process 數量大增的情況下 IO 時間會拉很長，所以要找到一個平衡。

Compare scalability. Does your program scale well? Why or why not? How can you achieve better scalability? You may discuss the two implementations separately or together.

我認為我的 program scale 的不夠好，雖然再 single node 的 scalability 圖表現的不差，但根據 multi-node 的圖，在 16 process 以上就幾乎沒有優化。如果要達到更高 scalability，可能要去改善伺服器的 IO 狀態。

Others :

我原版的程式在跑 odd-even sort 的迴圈時，是沒有在最後用 allreduce 得知大家的狀態，而是用 for 迴圈跑整個大 array 的 size，也就是 odd-even sort 的 worst case iteration 數量，因為我認為與其每一次都用 allreduce 造成 communication overhead，不如讓每一輪快一點多做幾輪，最後我發現跟 cpu_time 比，communication_time 根本不是問題，所以用 allreduce 快了大約 13%。

Experiences / Conclusion:

這次作業讓我第一次實作了自己的平行優化程式，我覺得非常好玩，也在測試實驗的過程中看到了自己的平行是真的有優化到。我原本認為 communication 會是 bottleneck 所以盡量避免了溝通，但是後來發現 cpu time 才是真正的 bottlnecck，我遇到最大的困難是記憶體的管控，我很常沒有開好 array 的大小或錯誤 acccess 導致 runtime error，在平行程式內記憶體真的非常重要。