**Homework 1: Mandelbrot Set**

**108062313 黃允暘**

**Implementation**

1. **Data distribution**:

在本此作業中，我將每個Node視odd even sort的交換單位。因此要先將資料分配到每個Node當中。而為了使每個Node分配到的data數量相同，我的作法如下：假設有n筆資料、Node的數量為size，每個Node會先分配到n/size筆資料。考慮到會有n/size會有餘數的問題，假設餘數是r，我會是讓前r個Node多分配到一筆data。

算出每個Node要取多少資料後，就可以計算出要讀取的資料位置(offset)，並透過MPI\_File\_read\_at ()，讀取資料。而為了等一下Node之間的資料傳輸，在這邊也會一並計算左右兩旁的Node大小。

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

1. **Sorting**:

根據odd even sort的做法，Node之間的資料交換會有兩個phase:

Odd phase: index為奇數的Node，會與右邊的Node進行資料換。

Even phase: index為偶數的Node，會與右邊的Node進行資料換。

在sorting實作的部分，我有使用到兩種sorting的辦法，分別應用在Node內部、Node之間。

1. Node內部：在與別的Node進行odd even sort之前，每個Node會先對內部的資料進行sorting。而我在這邊用的方法是spreadsort。
2. Node之間：Node之間資料的交換效率，會大大影響程式的performance。我在這邊採用的是類似merge sort的方式。兩個Node會互相傳輸自己的資料給對方，而傳輸的方式是採用MPI\_Sendrecv()。而因為資料在Node內部就已經排序好了，所以在合併兩個排序好的陣列時，我們只需要不斷比較兩邊的第一個元素，把比較小的丟到新的大陣列，就能完成排序。

只要不斷執行odd phase, even phase，就可以完成全部資料的sorting。另外，假設有K個Node，最多只要交換k+1次就可以完成sorting。

1. **Optimization**:
2. Merge sort:

其實我最一開始的做法不是merge sort，而是將左邊Node的資料傳給右邊的Node，直接要右邊的Node在內部執行quicksort。這樣的作法完全浪費了一個Node的計算資源。用merge sort的話，就可以利用到雙方的計算資源。

另外，在merge sort的過程，我還有做調整，使sorting速度加快。如同之前所提到的，兩邊的Node都會將資料傳給對方，但兩邊的Node都不用對全部的資料進行sorting，只需要找滿自己size的資料數量就好。左邊的Node之需要找滿前半部的資料就好，而右邊的Node只需要找滿後半部的資料就好。

一張含有 文字 的圖片

自動產生的描述

1. Early stop:

根據odd even sort的演算法，只要在一個phase中，都沒有發生過sorting，就可以結束odd even sort。因為當兩的Node進行資料交會時，個別的資料都是已經經過sorting，因此只要「左邊Node的最後一筆資料」大於「右邊Node的第一筆資料」，這兩個Node就不用進行sorting。

根據此特性，我的sorting function會的回傳值代表該次Node的交流，是否有發生sorting，1代表有，0代表沒有。而我在每個phase結束的時候，都會利用MPI\_Allreduce()，去檢查倆倆Node是否有發生sorting。若都沒發生的話，就會直接跳離迴圈，結束odd even sort。

1. 善用指標：

在執行merge sort的時候，有些步驟會需要將一個陣列指定給另一個陣列，如果使用for迴圈，複製陣列的內容，會花非常多的時間。因此用指標代替陣列的複製，會節省蠻多時間的。

**Experiment & Analysis**

1. **Methodology**
2. System spec: 學校的系統環境

一張含有 文字 的圖片

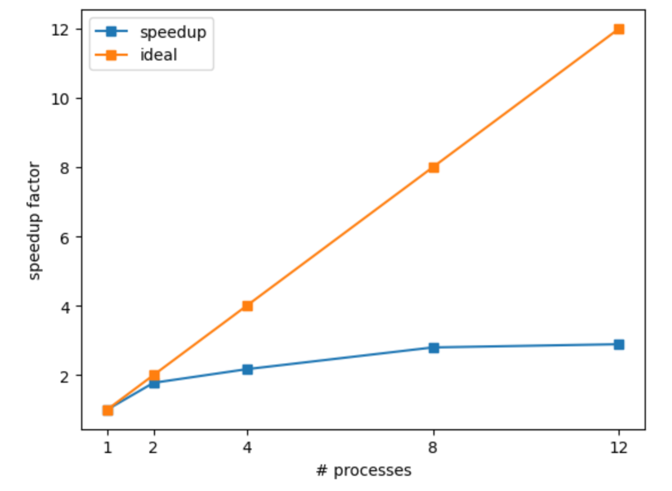
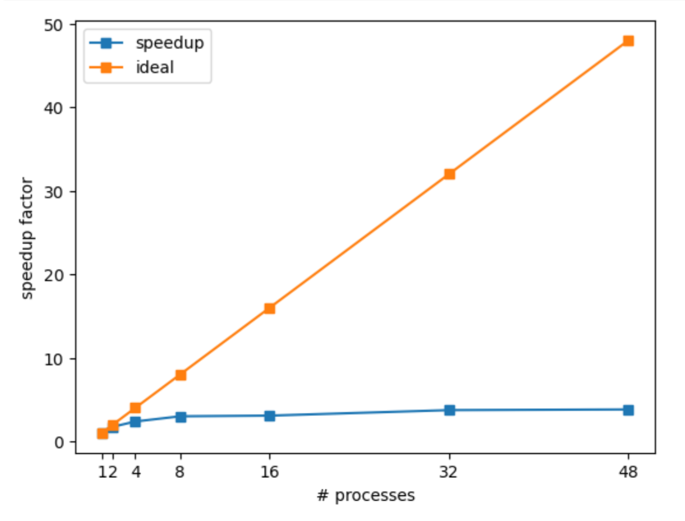
自動產生的描述

1. Performance Metrics:

我利用MPI\_Wtime()去計算程式執行的時間，計算的方式如下圖。

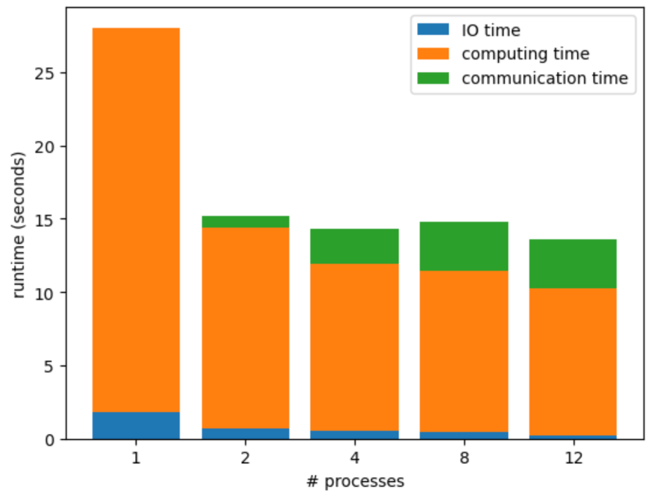
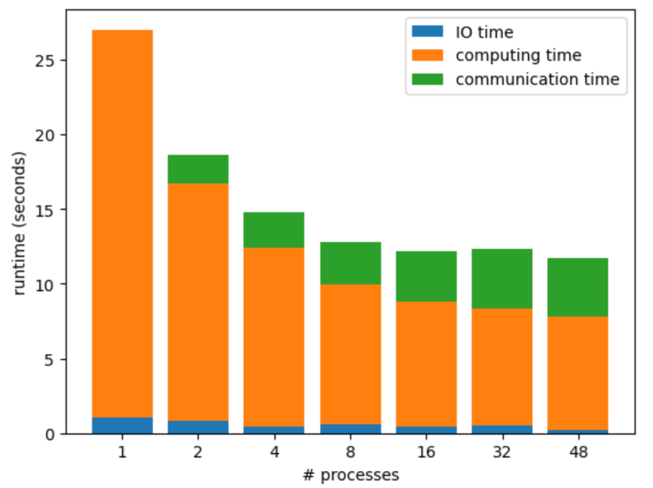
一張含有 文字 的圖片

自動產生的描述

1. **Speedup Factor & Time Profile**
2. Speedup Factor

(i)1 Node (ii)4 Nodes (12 processes/Node)

從圖表上可以看到，當processes的數量增多的時候，speedup factor確實有上升。但隨著processes的增加，speedup的程度離ideal越來越遠了。這是因為雖然processes的數量增加了，但communication的tradeoff也增加了，這一點可以從下一個實驗看出來。所以可以看到，當processes的數量從1增加到8時，speedup有明顯的上升; 但當processes從32升到48時，speedup卻幾乎沒有上升，這就代表communication time也上升的非常多。

1.  Time Profile

(i) 1 Node (ii)4 Nodes (12 processes/Node)

可以看到computing time隨著processes的增加，有了明顯的下降，這是因為有多個processes可以同時處理data。而隨著processes繼續增加，communication time 也有明顯的增加，因為隨著processes的增加，需要溝通的次數也增加了。這個現象也符合上一個speedup factor實驗的結論。

1. **Discussion**

透過實驗，可以發現雖然多個processes可以有效幫助computing time下降，但也會使communication time上升。因此processes並不是越多越好。另外，透過圖表也可以發現：computing time並不會隨著processes的數量上升，而成比例下降。我覺得這是因為程式的scalability仍有上升的空間，目前的程式沒有辦法完全利用多個processes的優勢，這是未來可以改進的地方。

**Experiences / Conclusion**

這次的實作使我更了解computing time、IO time 還有communication time的關係，也更了解MPI的操作。整體來說，實作odd even sort並不困難，困難的事之後的優化，從sorting的設計、for loop的設計到變數的宣告等等，有很多要注意的地方。另外，cluster的使用人數也會大大的影響execution time，同一份code，在人多或人少的時候跑，會有非常大的差異。