Hw1 MPI			
學號: 109062233	姓名: 蘇裕恆	繳交時間: 10/19	

1 Implementation :

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

Overall , I use count how many element each block should have by

```
void count_off_all(int& offset , int& allocated , int rank , int new_size , int n){
    int gap = n / new_size;
    offset = rank * gap ;
    if(rank != new_size - 1){ // not the last one
        allocated = gap;
    }
    else{
        allocated = n - gap * (new_size - 1);
    }
}
```

(The offset above is where I should start reading at.)

And to prevent the situation in tesetcase 3 that the processor is greater than the size of array , I use the MPI_Group_range_excl function to have the extra Processor kicked away.

```
if(size > n ){
    MPI_Group orig_group, new_group;
    MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
    // printf("%d into the excl func \n", rank);
    // discard the extra
    // Remove all unnecessary ranks
    int ranges[3] = { n   , size - 1 , 1 };
    MPI_Group_range_excl(orig_group, 1, &ranges, &new_group);
    bool flag = (rank >= n) ;
    // Create a new communicator
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
    if(new_comm == MPI_COMM_NULL){
        if(flag){
            // printf("abort the rank  %d \n " , rank);
            MPI_Finalize();
            return 0;
        }
    }
}
```

And finally make it after numerous tries.

- How do you sort in your program?
  I use radix sort as the sorting algorithm as my program , since the time complexity for the radix sort is O(n) as the floating point follow the IEEE standard , which is better than the performance of std::sort.

```
static inline unsigned int FloatFlip(float input){
    unsigned int f = *(unsigned int *)&input;
    unsigned int mask = -int(f >> 31) | 0x80000000; // -1 = 0xFFFFFFFF
    return f ^ mask;
}
```

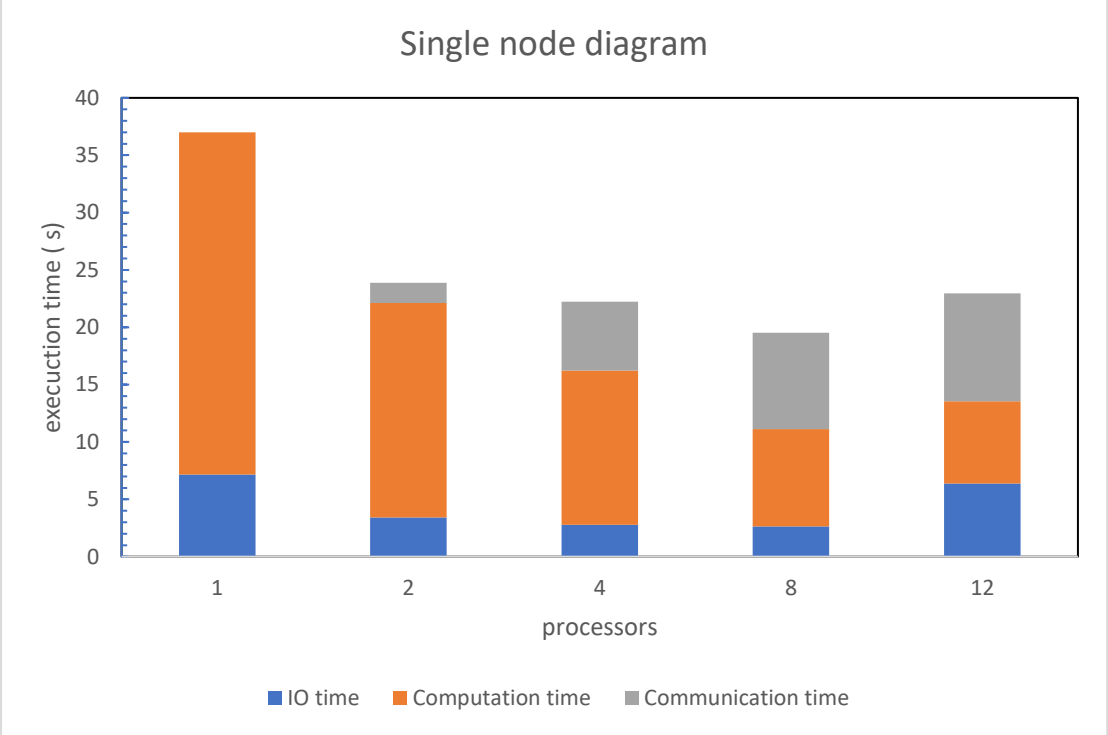
  The basic concept is that use the property of IEEE standard floating point to make the radix sort available

- Other efforts you've made in your program.
  I think most efforts are base on this two , since the if not doing so some data test wil not passed . And I think is time for my implementation is quite far from the tle.

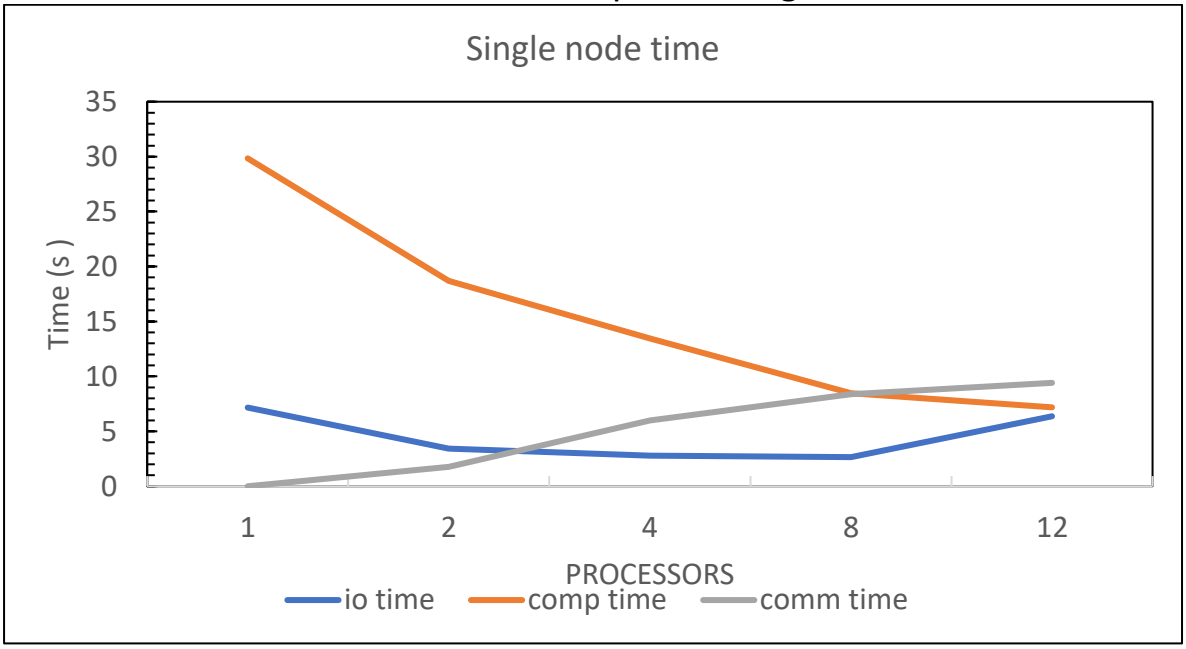
2 Experiment & Analysis

- Methodology :
  1. System Spec :
     Skip … I use Apollo as the working environment
  2. Performance Metrix
     I use the MPI_Wtime() which is presented in the lab1 to evaluate the performance of the time evaluation.
     And the way to separate the computing time / communication time and IO time in my program is that
     Computing time -> merge the two array O(n) and Radix sort at the beginning O(n) and other trivial calculation time.
     Communication time -> the SendRecv and the Allreduce function
     IO time -> the File_open , read and write at the begin and the end of the process
- Plots :
  Speedup Factor & Time Profile : In the following , I will use the testcase 38 which contain the testing data of size 536831999

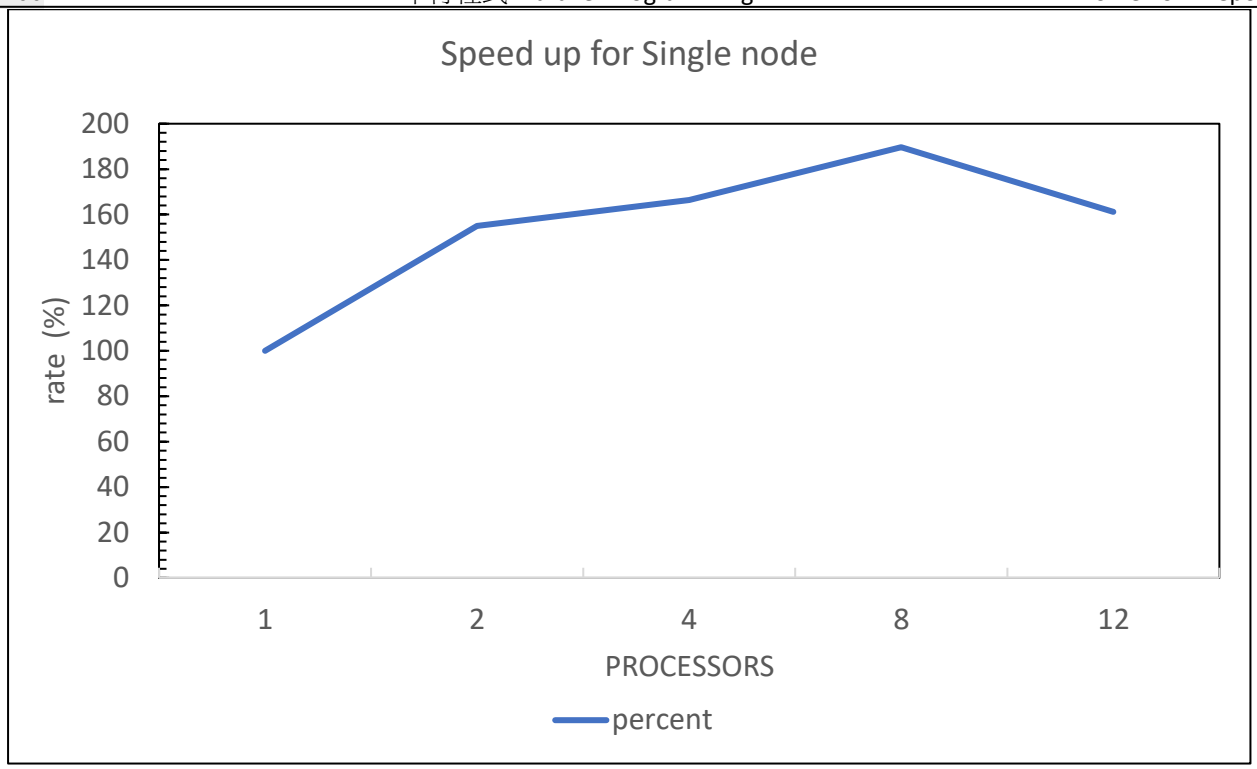
First , we start from the single node :



We can find that the performance get the best when there's 8 processors.
As the processors get more , the IO time get first drop then get larger and the communication time also increase as processor get more.

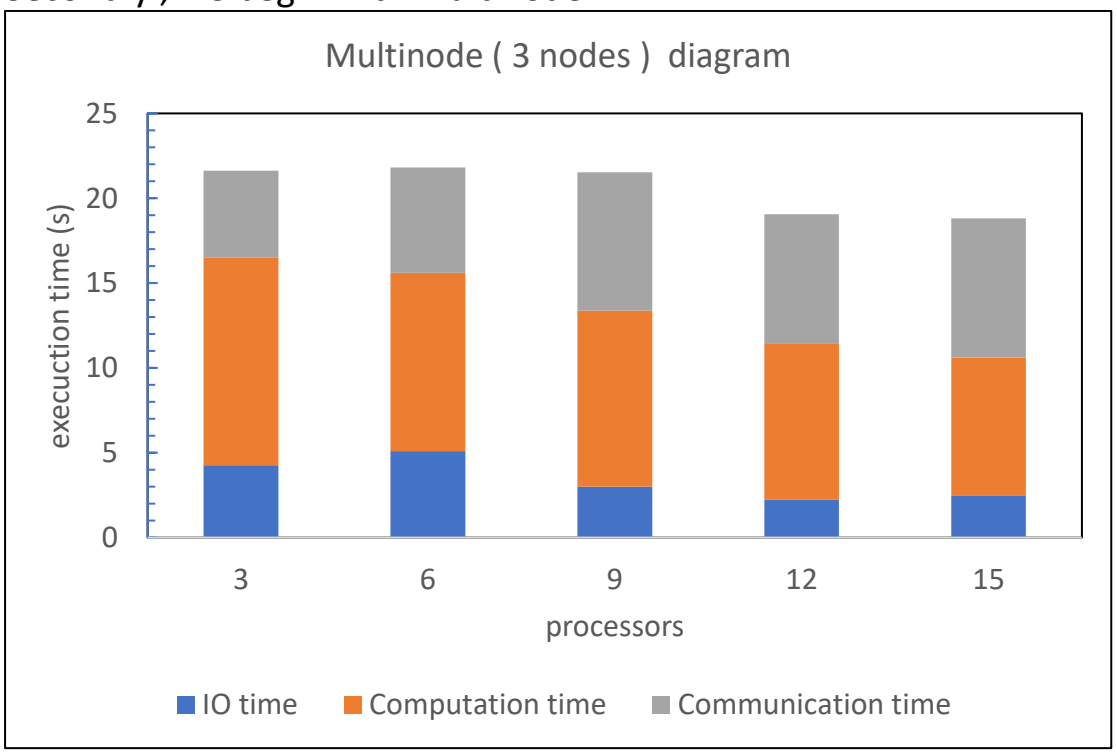


We can note that the even if the computation time for the process decrease long as the processors increase , other time increase like the communication time.
But what's surprised is that the io time increase as the processors exceed the threshold of 8. Guessing that it's similar to the concept of context switch in the operating system that make the performance down.

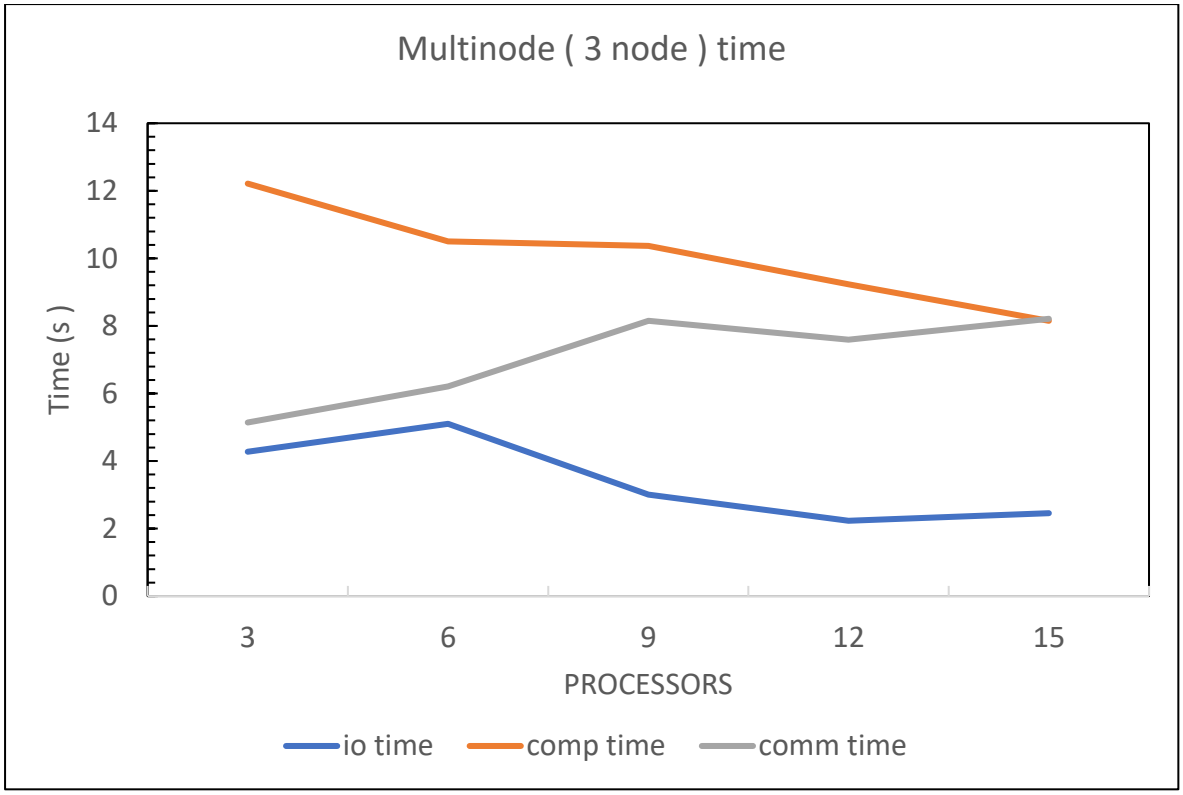


And the diagram above is the speed up for single node.
(For the processor is since the i/o time cost more , so the speedup drop)

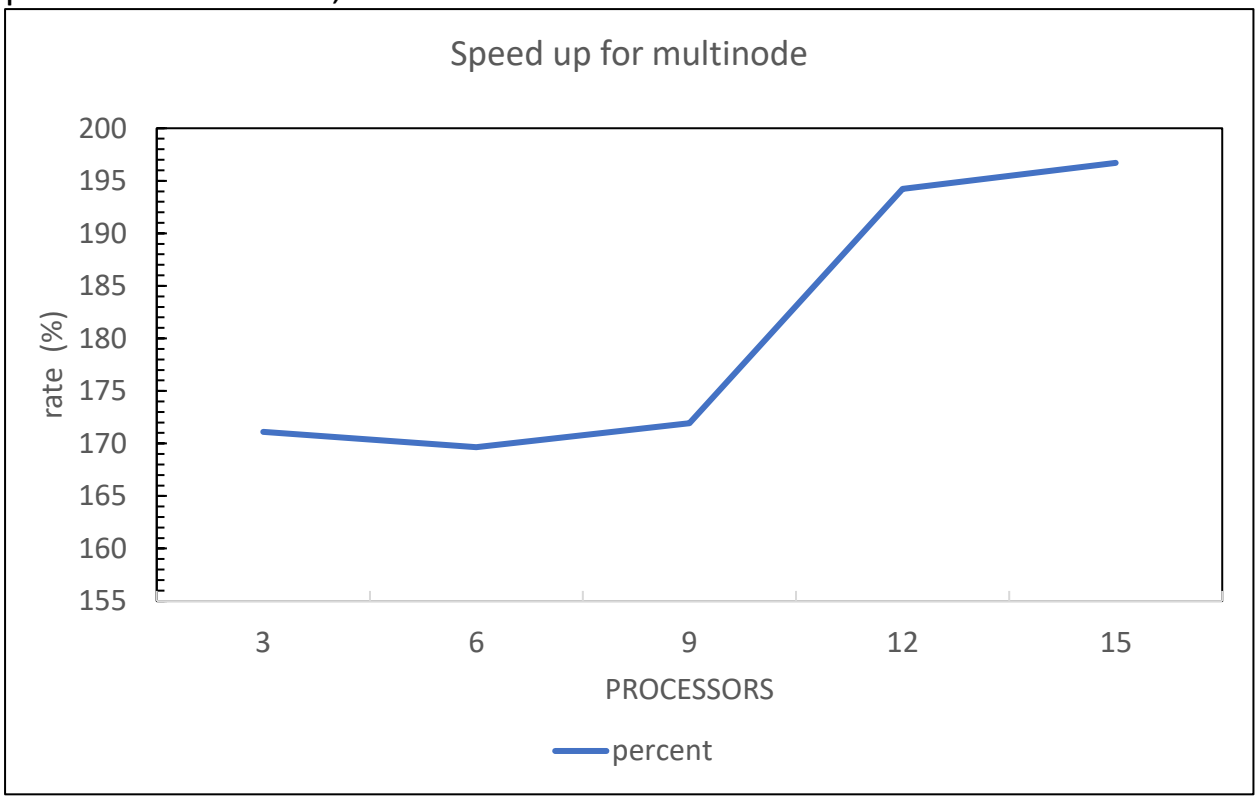
Secondly , we begin with multinode :



We can notice that the performance doesn't drop as the processor get 15. (Similar to 5 processors in the single node) And I believe that the it will drop as well when there's too many processor in each processor. (But in fact it didn't drop until 36 processors and the Apollo has limited processors)



We can see that , only the communication time get increase when the processor increases , and others decreases.



With the accelerate near 200%

- Discussion :
  1. Compare I/O, CPU, Network performance. Which is/are the bottleneck(s)? Why? How could it be improved?
     Ans :
     By the above , for single node :
     The bottleneck is the IO time , maybe a faster cpu that can help.
     For multi nodes :
     The bottleneck is the communication time , currently the method that I have done is that change the communication sendrecv to each rank and process the data in the local data , I can't think of the way to decrease the communication time since the implementation of mine is the best way I can think of.
  2. Does my program scale well ? Why or why not ?
     Theoratical analysis :
     The process of radix sort for my implementation is O(3*n) which is the I believe not bad when there's only one processor. While I believe the best performance we can achieve is O(1*n) to finish the sort. In that case , the best performance is approximately 1/3 of the original which has the same value of those top students.
     And the time improvement of my project is around 200% faster , which I think mentain the 66% of theoratical speed up for my implementation and I think abtain the 70% of ideal performance is not bad at all,
     To have a better scalability , I believe that we can perform the trade of between space and time.

```
// 3 histograms on the stack:
const int kHist = 2048;
unsigned int b0[kHist * 3];

unsigned int *b1 = b0 + kHist;
unsigned int *b2 = b1 + kHist;
```

     Maybe if we have only 2 histograms on the stack , that will give a 33% speed up ideally. But it required 2 * 2 * 2 ** 16 space of floats.
- Others : I want to try if the speed of multiple nodes will go done as each nodes have more processors , but the Apollo doesn't have the resource to do so qAq.

  Also , as for improvement , I try to use the memcpy to replace the for loop copy and only allreduce when the odd round ends , but the two makes little benefits.

Conclusion :

1. MPI is extremely hard for me to debug as the information of MPI library on the internet are old and some server even shotdown ^^
2. The most difficult part of MPI is to know how it work , and the concept of parallel programming is expecially hard.

```
if (myrank == 0) then
   call MPI_SEND(b,1,MPI_REAL,1,0,MPI_COMM_WORLD,ierr)
   call MPI_RECV(a,1,MPI_REAL,0,0,MPI_COMM_WORLD,istat,ierr)
elseif (myrank == 1) then
   call MPI_SEND(b,1,MPI_REAL,0,0,MPI_COMM_WORLD,ierr)
   call MPI_RECV(a,1,MPI_REAL,1,0,MPI_COMM_WORLD,istat,ierr)
end if
```

   Like the above will cause the deadlock and sendrcev will not, which is differ from what I've learned in the sequencial code
3. Feedback : Maybe it's possible to have our discussion on the 討論區 , I don't think it a good idea to always discuss by email and the environment for the Apollo is unstable , sometimes the test case may have change from 40 seconds to 20 seconds . (And I think the above discussion may be affect by this)