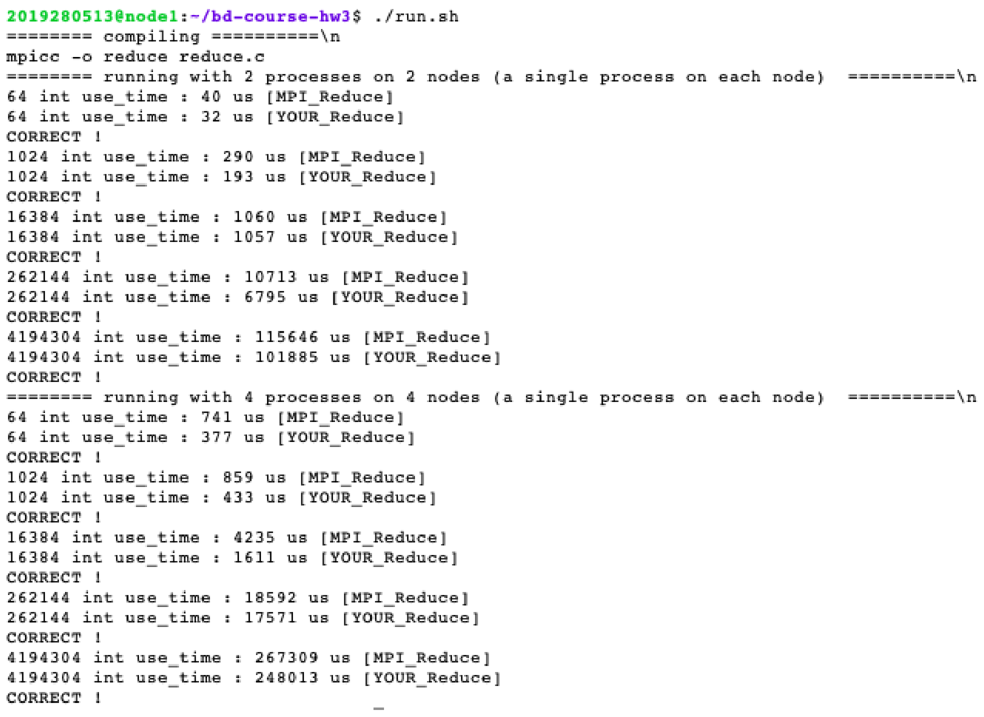
Sami Emre Erdogan(夏承思)
Student ID: 2019280513

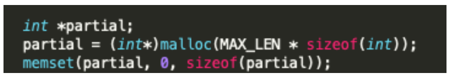
BIG DATA COURSEWORK 3
MPI PROGRAMMING



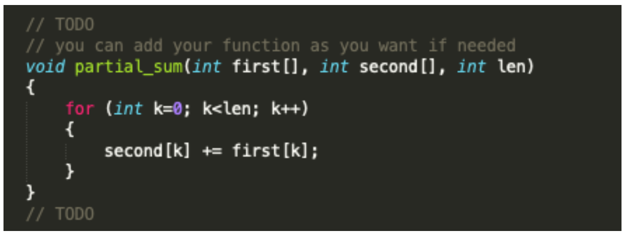
```
2019280513@node1:~/bd-course-hw3$ ./run.sh
======== compiling ==========\n
mpicc -o reduce reduce.c
======== running with 2 processes on 2 nodes (a single process on each node)  ==========\n
64 int use_time : 40 us [MPI_Reduce]
64 int use_time : 32 us [YOUR_Reduce]
CORRECT !
1024 int use_time : 290 us [MPI_Reduce]
1024 int use_time : 193 us [YOUR_Reduce]
CORRECT !
16384 int use_time : 1060 us [MPI_Reduce]
16384 int use_time : 1057 us [YOUR_Reduce]
CORRECT !
262144 int use_time : 10713 us [MPI_Reduce]
262144 int use_time : 6795 us [YOUR_Reduce]
CORRECT !
4194304 int use_time : 115646 us [MPI_Reduce]
4194304 int use_time : 101885 us [YOUR_Reduce]
CORRECT !
======== running with 4 processes on 4 nodes (a single process on each node)  ==========\n
64 int use_time : 741 us [MPI_Reduce]
64 int use_time : 377 us [YOUR_Reduce]
CORRECT !
1024 int use_time : 859 us [MPI_Reduce]
1024 int use_time : 433 us [YOUR_Reduce]
CORRECT !
16384 int use_time : 4235 us [MPI_Reduce]
16384 int use_time : 1611 us [YOUR_Reduce]
CORRECT !
262144 int use_time : 18592 us [MPI_Reduce]
262144 int use_time : 17571 us [YOUR_Reduce]
CORRECT !
4194304 int use_time : 267309 us [MPI_Reduce]
4194304 int use_time : 248013 us [YOUR_Reduce]
CORRECT !
```

To reduce the time complexity of the program, parallel execution of sub-arrays is being used by parallel programming MPI to calculate their partial sums and then finally, the master (root) which in our case is Array 0 calculates the sum of these partial sums and returns the total sum of the array.

In order to use partial sum in our code first we had to initialize variables and create a function that handles the partial summation for us.



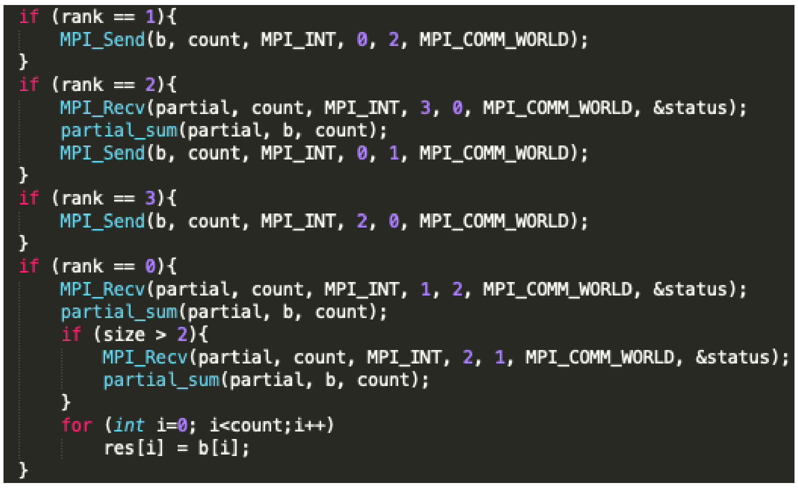
```
int *partial;
partial = (int*)malloc(MAX_LEN * sizeof(int));
memset(partial, 0, sizeof(partial));
```



```
// TODO
// you can add your function as you want if needed
void partial_sum(int first[], int second[], int len)
{
    for (int k=0; k<len; k++)
    {
        second[k] += first[k];
    }
}
// TODO
```

Sami Emre Erdogan(夏承思)
Student ID: 2019280513

In my MPI implementation. I have used MPI_Send and MPI_Recv functions. I also used 4 rank numbers which one of them was used for partial summation I tried to implement the code in such a way that to make sure that the IDs matched between the sender and the receiver, used status to make sure that the receiver successfully received the package the reason I have used size is >2 in rank 0 is that to make sure it was the last step to calculate the summation from other ranks and gather it in our root process. The for loop at the end of the screenshot also made sure that the master process can add its own array so that the program doesn't crash. Maybe there was a more efficient way to develop a better program by sending b from one process to another by multiplying by its size but I couldn't manage to optimize it any further but still my MPI program worked and performed slightly better. So, I am satisfied with the result.



```
if (rank == 1){
    MPI_Send(b, count, MPI_INT, 0, 2, MPI_COMM_WORLD);
}
if (rank == 2){
    MPI_Recv(partial, count, MPI_INT, 3, 0, MPI_COMM_WORLD, &status);
    partial_sum(partial, b, count);
    MPI_Send(b, count, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
if (rank == 3){
    MPI_Send(b, count, MPI_INT, 2, 0, MPI_COMM_WORLD);
}
if (rank == 0){
    MPI_Recv(partial, count, MPI_INT, 1, 2, MPI_COMM_WORLD, &status);
    partial_sum(partial, b, count);
    if (size > 2){
        MPI_Recv(partial, count, MPI_INT, 2, 1, MPI_COMM_WORLD, &status);
        partial_sum(partial, b, count);
    }
    for (int i=0; i<count;i++)
        res[i] = b[i];
}
```