Parallel Computing Lab
Nilay Ganvit - 200001053
1st September 2022

Lab 1
Merge Sort
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"

void exchange(int *a,int *b){

  int temp=*a;
  *a=*b;
  *b=temp;

}

int partition(int arr[],int begin,int fin){

    int p=arr[begin];

    int cnt=0;
    for(int i=begin+1;i<=fin;i++){
        if(arr[i]<=p){
            cnt++;
        }
    }

    int p_ind=begin+cnt;
    exchange(&arr[p_ind],&arr[begin]);
    int i=begin,j=fin;

    while(i< p_ind&&j>p_ind){

        while(arr[i]<=p){
            i++;
        }
```

```
        while(arr[j]>p){
            j--;
        }

        if(i<p_ind&&j>p_ind){
            exchange(&arr[i++], &arr[j--]);
        }
    }

    return p_ind;
}

void quickSort(int arr[],int begin,int fin){

    if(begin>=fin){
        return;
    }
    int p=partition(arr,begin,fin);
    quickSort(arr,begin,p-1);
    quickSort(arr,p+1,fin);
}

void merge(int arr[],int lower,int upper,int len){

    int lencurr=(upper-lower+1);

    if(lencurr==len){
        return;

    }

    int m=(lower+upper)/2;

    merge(arr,lower,m,len);
    merge(arr,m+1,upper,len);

    int n1=m-lower+1;
    int n2=upper-m;
```

```
    int L[n1],M[n2];

    for(int i=0;i<n1;i++){
        L[i]=arr[lower+i];
    }
    for(int j=0;j<n2;j++){
        M[j]=arr[m+1+j];
    }

    int i,j,k;
    i=0;
    j=0;
    k=lower;

    while(i<n1&&j<n2){
        if(L[i]<=M[j]){
            arr[k]=L[i];
            i++;
        }else{
            arr[k]=M[j];
            j++;
        }
        k++;
    }

    while(i<n1){
        arr[k]=L[i];
        i++;
        k++;
    }

    while(j<n2){
        arr[k]=M[j];
        j++;
        k++;
    }

}

int main(int argc, char **argv){
```

```
    int np,pid,work_num,source,dest,lower,i,cnt=0,N,batch_size;

    N = atoi(argv[1]);

    int arr[N];

    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&pid);
    MPI_Comm_size(MPI_COMM_WORLD,&np);

    work_num=np;

    if(pid==0){

        for(int i=0;i<N;++i){
            int a;
            scanf("%d",&a);
            arr[i]=a;
        }
        printf("Input: ");
        for(int i=0;i<N;i++){
                printf("%d ",arr[i]);
        }
        printf("\n");

        clock_t begin=clock();

        lower=0;
        batch_size=N/work_num;

        for(dest=1;dest<work_num;dest++){

            MPI_Send(&lower, 1,MPI_INT,dest,1,MPI_COMM_WORLD);
            MPI_Send(&batch_size,1,MPI_INT,dest,1,MPI_COMM_WORLD);
            MPI_Send(&arr[lower],batch_size,MPI_INT,dest,1,
MPI_COMM_WORLD);
            lower+=batch_size;
```

```
        }

        quickSort(arr,lower,lower+batch_size-1);

        for(i=1;i<work_num;i++){

            source=i;
            MPI_Recv(&lower,1,MPI_INT,source,2,MPI_COMM_WORLD,&status);

MPI_Recv(&batch_size,1,MPI_INT,source,2,MPI_COMM_WORLD,&status);

MPI_Recv(&arr[lower],batch_size,MPI_INT,source,2,MPI_COMM_WORLD,&status);

        }

        merge(arr,0,N-1,batch_size);

        printf("Sorted array: ");

        for(int i=0;i<N;i++){
            printf("%d ",arr[i]);
        }
        printf("\n");

        clock_t fin = clock();

        printf("Done\n");
        printf("Time taken : %lf sec \n",
((double)(fin-begin)/CLOCKS_PER_SEC));
  }

  if(pid>0){

    source=0;

    MPI_Recv(&lower,1,MPI_INT,source,1,MPI_COMM_WORLD,&status);
    MPI_Recv(&batch_size,1,MPI_INT, source,1,MPI_COMM_WORLD,&status);
    MPI_Recv(&arr,batch_size,MPI_INT,source,1,MPI_COMM_WORLD,&status);

    quickSort(arr,0, batch_size-1);
```

```
    MPI_Send(&lower,1,MPI_INT,0,2,MPI_COMM_WORLD);
    MPI_Send(&batch_size,1,MPI_INT,0,2,MPI_COMM_WORLD);
    MPI_Send(&arr,batch_size,MPI_INT,0,2,MPI_COMM_WORLD);
  }

  MPI_Finalize();
}
```

Input/Output & Time Taken:

```
nilay@Nilay-PC:~$ mpicc -o mpi lab3.c -lm
nilay@Nilay-PC:~$ mpiexec -n 6 ./mpi 16
1
5
9
3
5
7
0
2
4
6
8
12
45
78
32
65
Input: 1 5 9 3 5 7 0 2 4 6 8 12 45 78 32 65
Sorted array: 0 1 2 3 4 5 5 6 7 8 9 12 32 45 65 78
Done
Time taken : 0.000597 sec
nilay@Nilay-PC:~$
```