APURV TODKAR
16BCB0048

LAB EXPERIMENT 3

PARALLEL AND DISTRIBUTED COMPUTING

1.Write a MPI code to implement quick sort .

CODE:

```
/* quicksort */
#include <stdio.h>
#include <mpi.h>
include <time.h>

#define N 1000000

void showElapsed(int id, char *m);
void showVector(int *v, int n, int id);
```

```
int * merge(int *v1, int n1, int *v2, int n2);
oid swap(int *v, int i, int j);
void qsort(int *v, int left, int right);

double startTime, stopTime;

void showElapsed(int id, char *m)
{
                    printf("%d: %s %f secs\n",id,m,(clock()-
                            startTime)/CLOCKS_PER_SEC);
}

void showVector(int *v, int n, int id)
{
        int i;
        printf("%d: ",id);
        for(i=0;i<n;i++)
                printf("%d ",v[i]);
putchar('\n');
}
int * merge(int *v1, int n1, int *v2, int n2)
{
        int i,j,k;
        int * result;

        result = (int *)malloc((n1+n2)*sizeof(int));

        i=0; j=0; k=0;
        while(i<n1 && j<n2)
                if(v1[i]<v2[j])
                {
                        result[k] = v1[i];
```

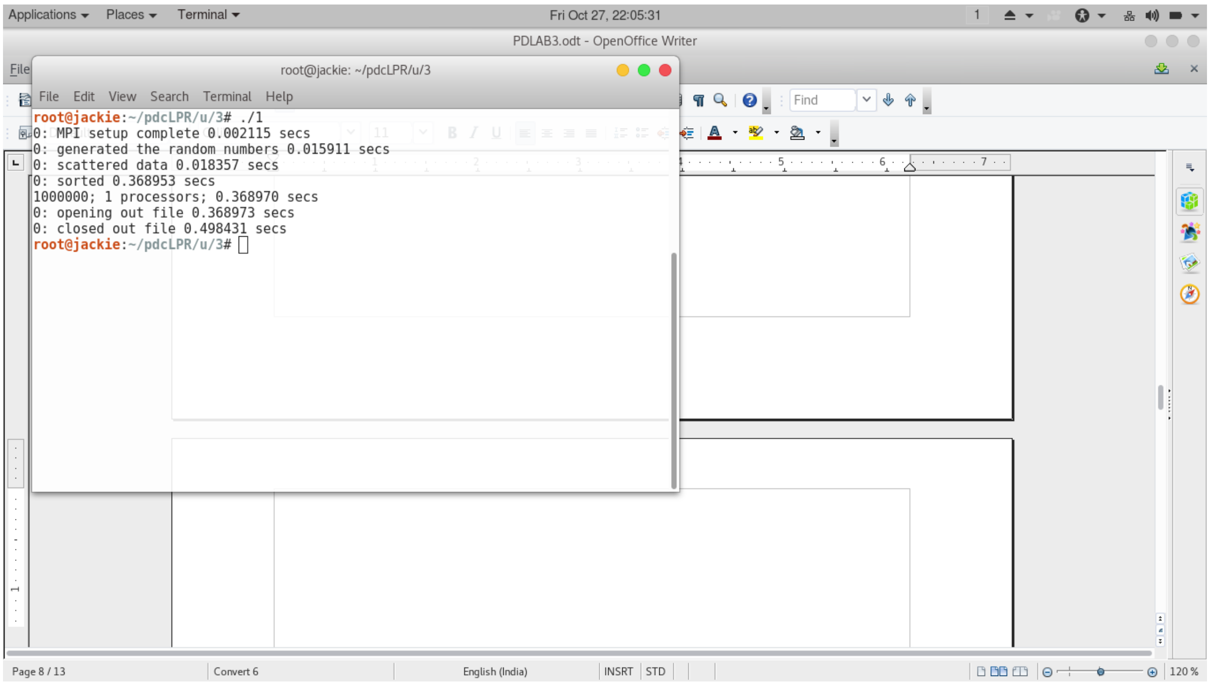
```
				i++; k++;
			}
			else
			{
				result[k] = v2[j];
				j++; k++;
			}
		if(i==n1)
			while(j<n2)
			{
				result[k] = v2[j];
				j++; k++;
			}
		else
			while(i<n1)
			{
				result[k] = v1[i];
				i++; k++;
			}
		return result;
}

oid swap(int *v, int i, int j)
{
int t;
		t = v[i];
		v[i] = v[j];
		v[j] = t;
}

void qsort(int *v, int left, int right)
```

```
{
	int i,last;
	if(left>=right)
		return;
	swap(v,left,(left+right)/2);
	last = left;
	for(i=left+1;i<=right;i++)
		if(v[i]<v[left])
			swap(v,++last,i);
	swap(v,left,last);
	qsort(v,left,last-1);
	qsort(v,last+1,right);
}

main(int argc, char **argv)
{
	int * data;
	int * chunk;
	int * other;
	int m,n=N;
	int id,p;
	int s;
	int i;
	int step;
	MPI_Status status;

	startTime = clock();

	MPI_Init(&argc,&argv);
	MPI_Comm_rank(MPI_COMM_WORLD,&id);
	MPI_Comm_size(MPI_COMM_WORLD,&p);
```

```
	showElapsed(id,"MPI setup complete");

	if(id==0)
	{
		int r;
		srandom(clock());

		s = n/p;
		r = n%p;
		data = (int *)malloc((n+s-r)*sizeof(int));
		for(i=0;i<n;i++)
			data[i] = random();
		if(r!=0)
		{
			for(i=n;i<n+s-r;i++)
				data[i]=0;
			s=s+1;
		}
		showElapsed(id,"generated the random numbers");

		MPI_Bcast(&s,1,MPI_INT,0,MPI_COMM_WORLD);
		chunk = (int *)malloc(s*sizeof(int));
MPI_Scatter(data,s,MPI_INT,chunk,s,MPI_INT,0,MPI_COMM_WORLD);

		showElapsed(id,"scattered data");

		qsort(chunk,0,s-1);

		showElapsed(id,"sorted");
	}
```

```
else
{
        MPI_Bcast(&s,1,MPI_INT,0,MPI_COMM_WORLD);
        chunk = (int *)malloc(s*sizeof(int));
        MPI_Scatter(data,s,MPI_INT,chunk,s,MPI_INT,0,MPI_COMM_WORLD);

        showElapsed(id,"got data");

        qsort(chunk,0,s-1);

        showElapsed(id,"sorted");
}

step = 1;
while(step<p)
{
        if(id%(2*step)==0)
        {
                if(id+step<p)
                {
                        MPI_Recv(&m,1,MPI_INT,id+step,0,MPI_COMM_WORLD,&
                        status); other = (int *)malloc(m*sizeof(int));

MPI_Recv(other,m,MPI_INT,id+step,0,MPI_COMM_WORL
                D,&status); showElapsed(id,"got merge
                data");
                        chunk = merge(chunk,s,other,m);
                showElapsed(id,"merged data");
                        s = s+m;
                }
}
        else
```

```
	{
			int near = id-step;
			MPI_Send(&s,1,MPI_INT,near,0,MPI_COMM_WORLD);
		MPI_Send(chunk,s,MPI_INT,near,0,MPI_COMM_WORLD);
			showElapsed(id,"sent merge data");
		break;
		}
		step = step*2;
	}
	if(id==0)
{
		FILE * fout;

	stopTime = clock();
				printf("%d; %d processors; %f secs\n", s,p,(stopTime-
							startTime)/CLOCKS_PER_SEC);

	showElapsed(id,"opening out file");
		fout = fopen("result","w");
		for(i=0;i<s;i++)
		fprintf(fout,"%d\n",chunk[i]);
		fclose(fout);
		showElapsed(id,"closed out file");
	}
	MPI_Finalize();
}
```


Applications
Places
Terminal
Fri Oct 27, 22:05:31
PDLAB3.odt - OpenOffice Writer
root@jackie: ~/pdcLPR/u/3
File Edit View Search Terminal Help
root@jackie:~/pdcLPR/u/3# ./1
0: MPI setup complete 0.002115 secs
0: generated the random numbers 0.015911 secs
0: scattered data 0.018357 secs
0: sorted 0.368953 secs
1000000; 1 processors; 0.368970 secs
0: opening out file 0.368973 secs
0: closed out file 0.498431 secs
root@jackie:~/pdcLPR/u/3#
Find
Page 8 / 13
Convert 6
English (India)
INSRT
STD
120 %

2. Write a parallel program using MPI to sort a given set of integers using merge sort.

CODE:

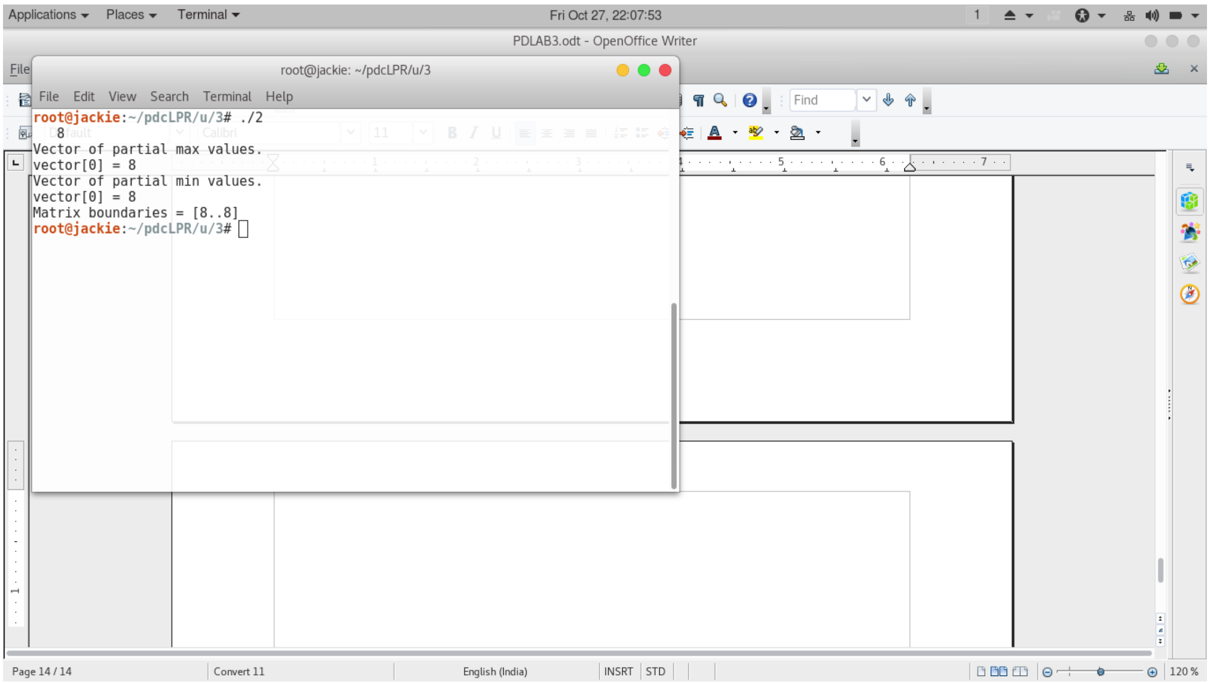
```
#include <mpi.h>

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

const int n = 1;
const int m = 1;

void fill_matrix_randomly(int matrix[n][m], int
max_value); void write_matrix(int matrix[n]
[m]);
int find_max(int* vector, int vector_size);
int find_min(int* vector, int vector_size);
void write_vector(int* vector, int vector_size);

int main(int argc, char* argv[])
{
   int my_rank = 0;
   int comm_size = 0;

   int a[n][m];
  int receive_buffer[m];
   int partial_max[m];
   int partial_min[m];

   MPI_Init(&argc, &argv);
```

```
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  MPI_Comm_size(MPI_COMM_WORLD, &comm_size);


  if (comm_size != n)

  {

    printf("Please set process count = %d and run again.", n);

    MPI_Finalize();

    return 0;

  }


  if (my_rank == 0)

  {

    fill_matrix_randomly(a, 10);

    write_matrix(a);

  }

  /* MPI Scatter(address of send buffer, number of elements sent to each
process, data type of send buffer, address of receive buffer, number of elements
in receive buffer, data type of receive buffer, rank of sending process,
communicators space) */

  MPI_Scatter(a, n, MPI_INT, receive_buffer, n, MPI_INT, 0, MPI_COMM_WORLD);

  /* MPI_Reduce(address of send buffer, address of receive buffer, number of
elements in send buffer, data type of elements in send buffer, reduce operation,
rank of root process, communicators space) */

  MPI_Reduce(receive_buffer, partial_max, n, MPI_INT, MPI_MAX, 0,
  MPI_COMM_WORLD);

  MPI_Reduce(receive_buffer, partial_min, n, MPI_INT, MPI_MIN, 0,
  MPI_COMM_WORLD);

  if (my_rank == 0)

  {

    printf("Vector of partial max values.\n");

    write_vector(partial_max, n);

    printf("Vector of partial min values.\n");

    write_vector(partial_min, n);
```

```
    int max = find_max(partial_max, n);
    int min = find_min(partial_min, n);
    printf("Matrix boundaries = [%d..%d]\n", min, max);
  }
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();

  return 0;
}

int find_max(int* vector, int vector_size)
{
  int max = vector[0];
  int i = 0;
  for (i = 0; i < vector_size; i++)
  {
    if (vector[i] > max)
    {
      max = vector[i];
    }
  }
  return max;
}

int find_min(int* vector, int vector_size)

  int min = vector[0];
  int i = 0;
  for (i = 0; i < vector_size; i++)
  {
    if (vector[i] < min)
```

```
      {
        min = vector[i];
      }
    }
    return min;


void fill_matrix_randomly(int matrix[n][m], int max_value)
{
    int i = 0;
    int j = 0;
    srand(time(NULL));
    for (i = 0; i < n; i++)
    {
      for (j = 0; j < m; j++)
      {
        matrix[i][j] = rand() % max_value;
      }
    }
}
void write_matrix(int matrix[n][m])
{
    int i = 0;
    int j = 0;
    for (i = 0; i < n; i++)
    {
      for (j = 0; j < m; j++)
      {
        printf("%4d", matrix[i][j]);
      }
      printf("\n");
```

```
  }
}

oid write_vector(int* vector, int vector_size)
{
  int i = 0;
  for (i = 0; i < vector_size; i++)
  {
    printf("vector[%d] = %d\n", i, vector[i]);
  }
}
```


Applications Places Terminal
Fri Oct 27, 22:07:53
PDLAB3.odt - OpenOffice Writer
root@jackie: ~/pdcLPR/u/3
File Edit View Search Terminal Help
root@jackie:~/pdcLPR/u/3# ./2
8
Vector of partial max values.
vector[0] = 8
Vector of partial min values.
vector[0] = 8
Matrix boundaries = [8..8]
root@jackie:~/pdcLPR/u/3#
Find
Page 14 / 14
Convert 11
English (India)
INSRT
STD
120 %