**Name: Abinash Satapathy**

**Reg. no.: 16BCE0081**

**Slot: L45 + L46**

**Subject: Parallel and Distributed Computing (CSE4001) Lab**

**Experiment – 4**

1. **Write an MPI Program to perform binary search**

#include<stdio.h>

#include<time.h>

#include<mpi.h>

int main(int argc,char \*argv[])

{

clock\_t tic=clock();

int rank,size;

int a[10]={1,2,3,4,5,6,7,8,9,10},b[10];

int search=6,flag=0;

int i;

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

MPI\_Scatter(&a,5,MPI\_INT,&b,5,MPI\_INT,0,MPI\_COMM\_WORLD);

if (rank==0)

{

for(i=0;i<5;i++)

{

if(b[i]==search)

{

printf("\nNumber Found!\t\t%d\t\t%d",rank,i);

flag=1;

}

printf("\n%d\t\t%d",b[i],rank);

}

}

if(rank==1)

for(i=0;i<5;i++)

{

if(b[i]==search)

{

printf("\nNumber Found!\t\t%d\t\t%d",rank,i);

flag=1;

}

printf("\n%d\t\t%d",b[i],rank);

}

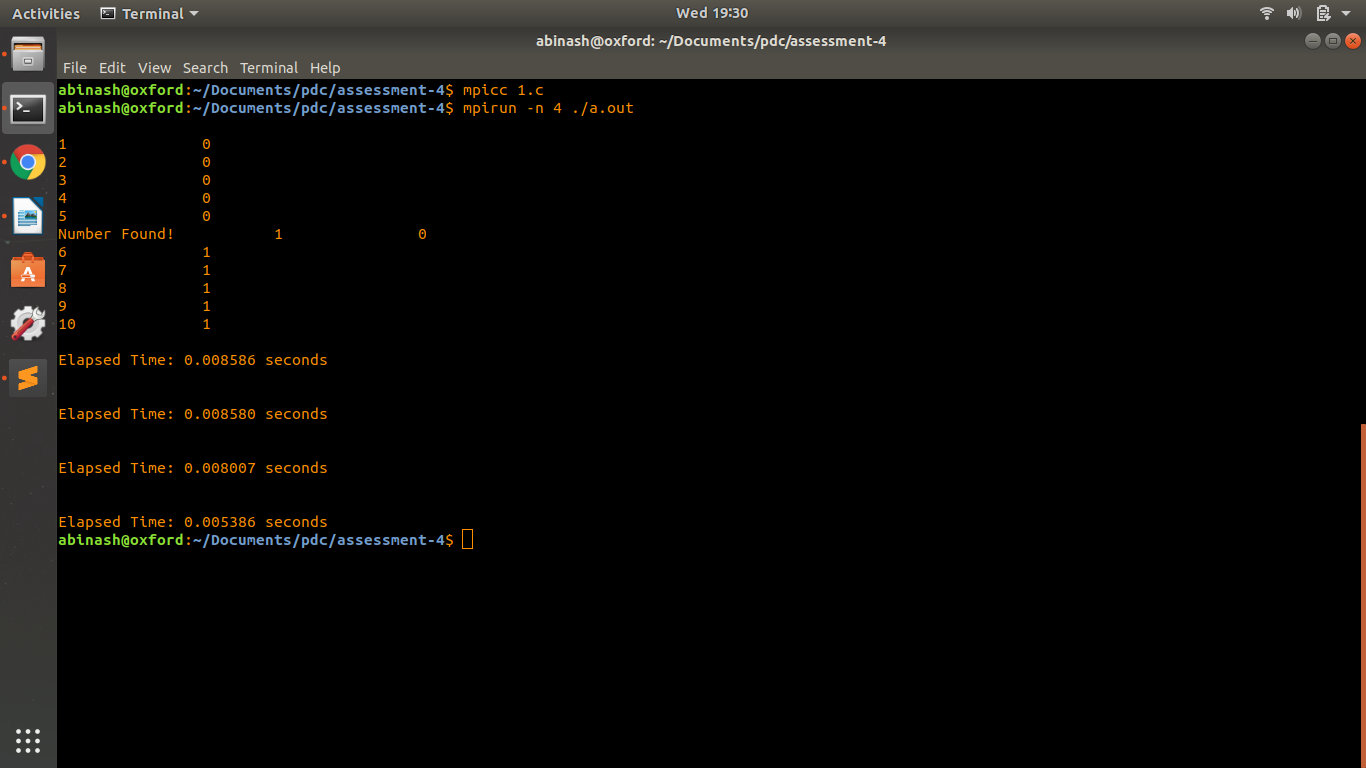
MPI\_Finalize();

clock\_t toc=clock();

printf("\n\nElapsed Time: %f seconds\n",(double)(toc-tic)/CLOCKS\_PER\_SEC);

return(0);

}



1. **Write an MPI program to perform ring communication**

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

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

MPI\_Init(NULL, NULL);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

int token;

if(world\_rank!=0){

MPI\_Recv(&token, 1, MPI\_INT, world\_rank-1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d received token %d from Process %d\n", world\_rank, token, world\_rank-1);

}

else

token = 1;

MPI\_Send(&token, 1, MPI\_INT, (world\_rank+1)%world\_size, 0, MPI\_COMM\_WORLD);

if(world\_rank==0){

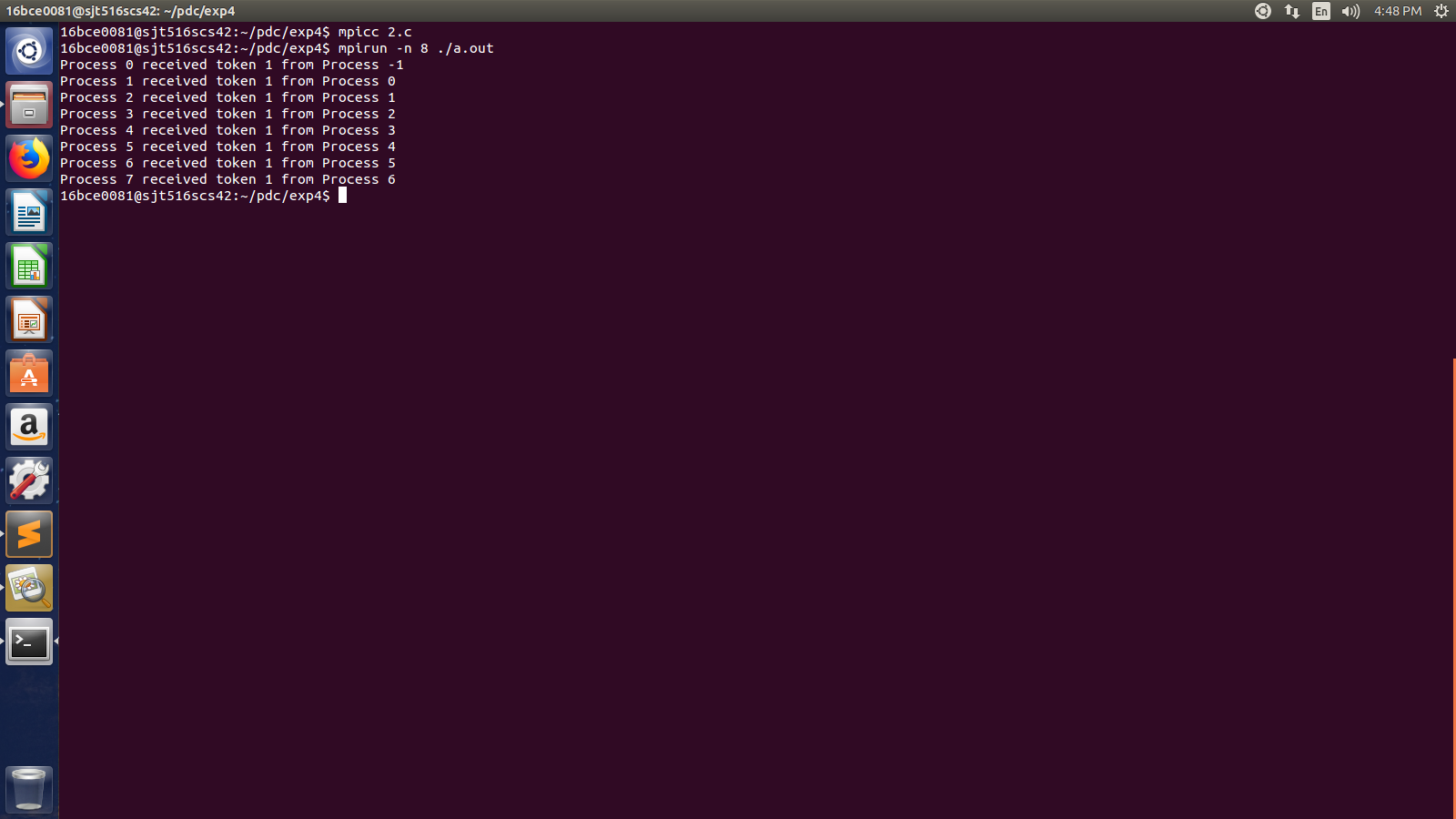
MPI\_Recv(&token, 1, MPI\_INT, world\_rank-1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d received token %d from Process %d\n", world\_rank, token, world\_rank-1);

}

return 0;

}



1. **Write an MPI program to perform the squaring of numbers in array.**

**Input sequence: 2 4 8 16**

**Output sequence: 4 16 64 256**

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#include <math.h>

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

int n, i;

int size, rank;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Barrier(MPI\_COMM\_WORLD);

if(rank==0){

int n, i;

printf("Enter the number of elements:\n");

scanf("%d", &n);

int arr[n];

printf("Enter the array:\n");

for(i=0;i<n;i++)

scanf("%d", &arr[i]);

printf("\n");

printf("The resultant array is:\n");

for(i=0;i<n;i++)

printf("%d ", (arr[i]\*arr[i]));

printf("\n");

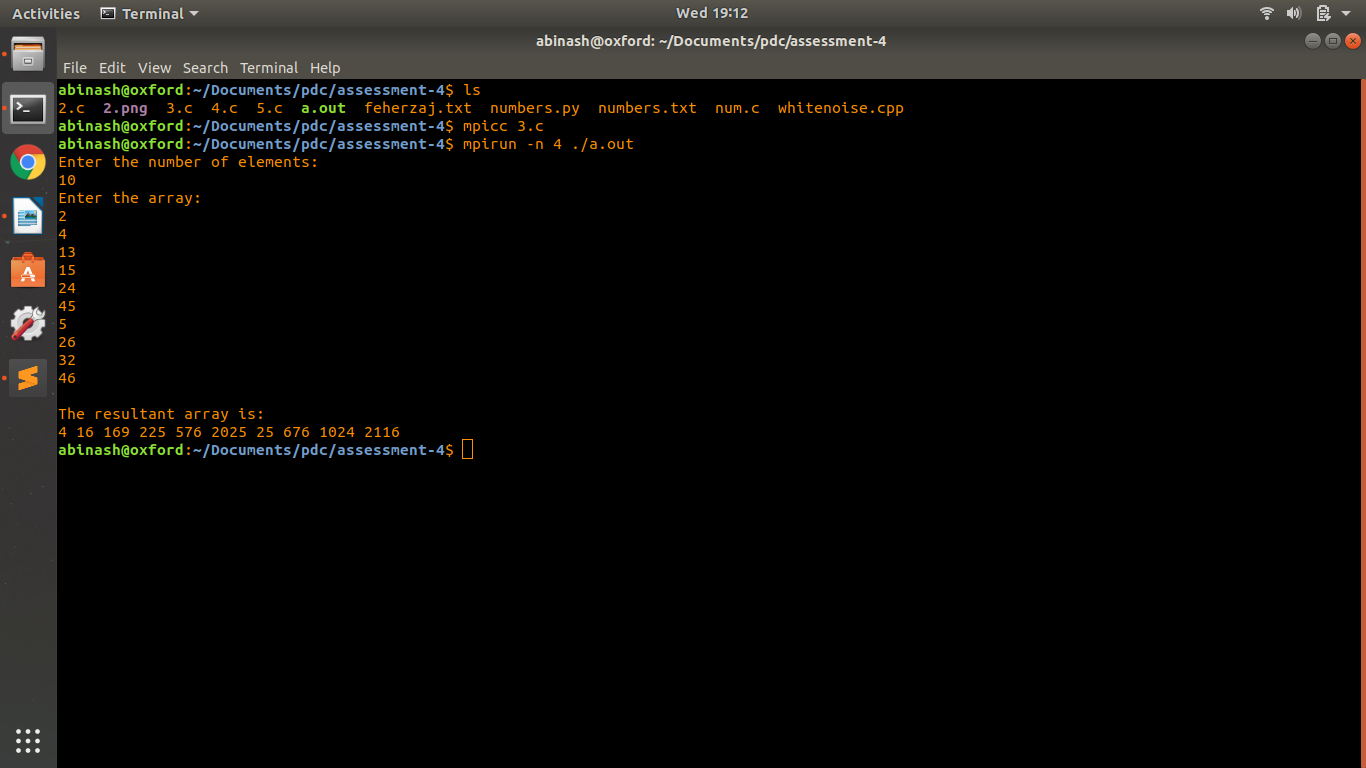
}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

return 0;

}



1. **Write an MPI program to perform the sum of 1000 numbers using gather and scatter.**

#include <stdio.h>

#include <mpi.h>

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

int i, j, k, p;

int a[1000], b[2500], c[4], myrank, res, x, y;

int interval, sum = 0;

for(i=0;i<1000;i++)

a[i] = i;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

MPI\_Scatter(a, 250, MPI\_INT, b, 250, MPI\_INT, 0, MPI\_COMM\_WORLD);

res = 0;

for(i=0;i<250;i++)

res = res + b[i];

MPI\_Gather(&res, 1, MPI\_INT, c, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

int final = 0;

if(myrank==0){

for(i=0;i<4;i++){

printf("The sum that is calculated in core %d is %d \n", (i+1), c[i]);

final = final + c[i];

}

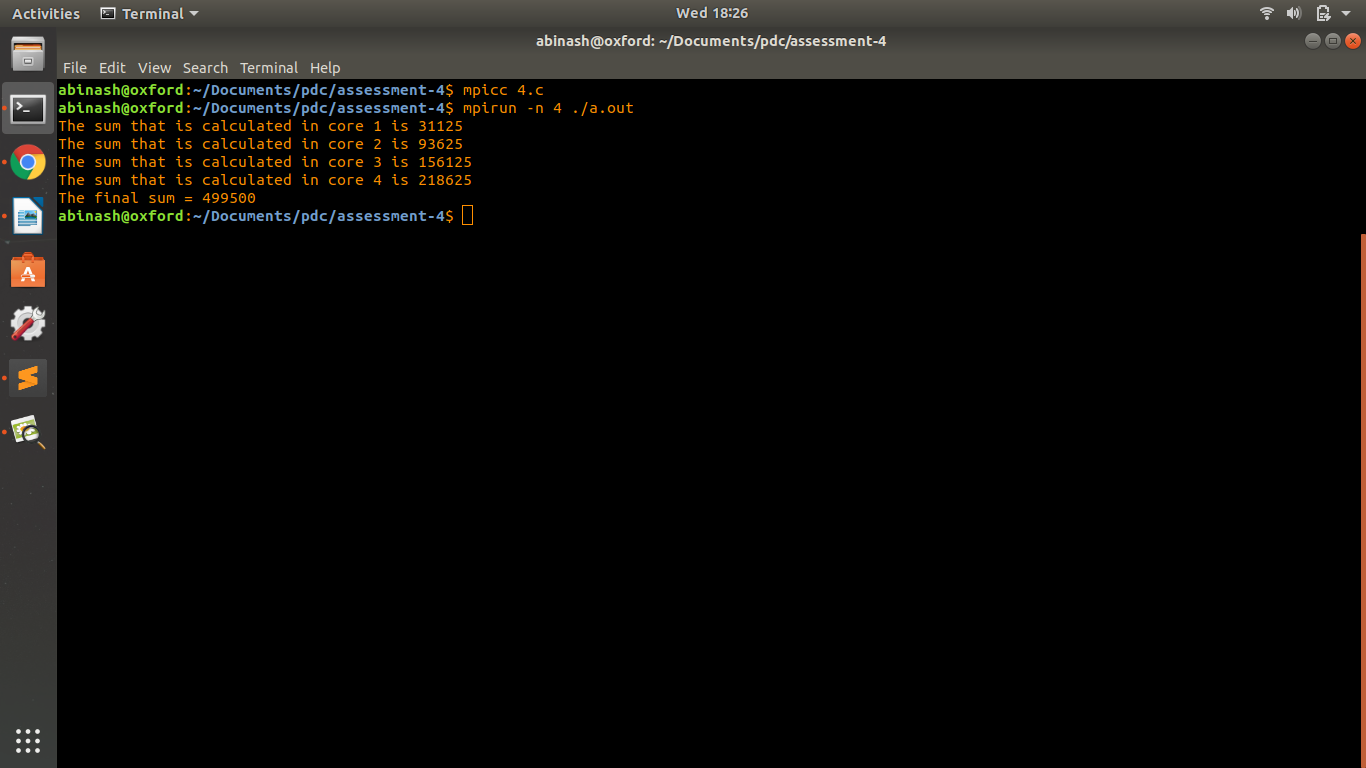
printf("The final sum = %d\n", final);

}

MPI\_Finalize();

return 0;

}



1. **Write a MPI program to perform the sum of 1000 numbers using MPI broadcast and reduce function. Calculate the time using MPI wall time function.**

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#define maxsize 1000

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

double t1, t2;

int myid, numprocs;

int low, high, myresult=0, result;

char fn[255];

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

t1 = MPI\_Wtime();

int i, x, data[1000];

FILE \*fptr;

fptr = fopen("numbers.txt", "r");

if(fptr==NULL){

printf("Error!\n");

exit(1);

}

for(i=0;i<1000;i++)

fscanf(fptr, "%d", &data[i]);

fclose(fptr);

MPI\_Bcast(data, maxsize, MPI\_INT, 0, MPI\_COMM\_WORLD);

x = maxsize/numprocs;

low = myid \* x;

high = low + x;

for(i=low;i<high;i++)

myresult = myresult + data[i];

printf("Obtained %d from %d\n",myresult, myid);

MPI\_Reduce(&myresult, &result, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if(myid==0)

printf("Sum = %d\n", result);

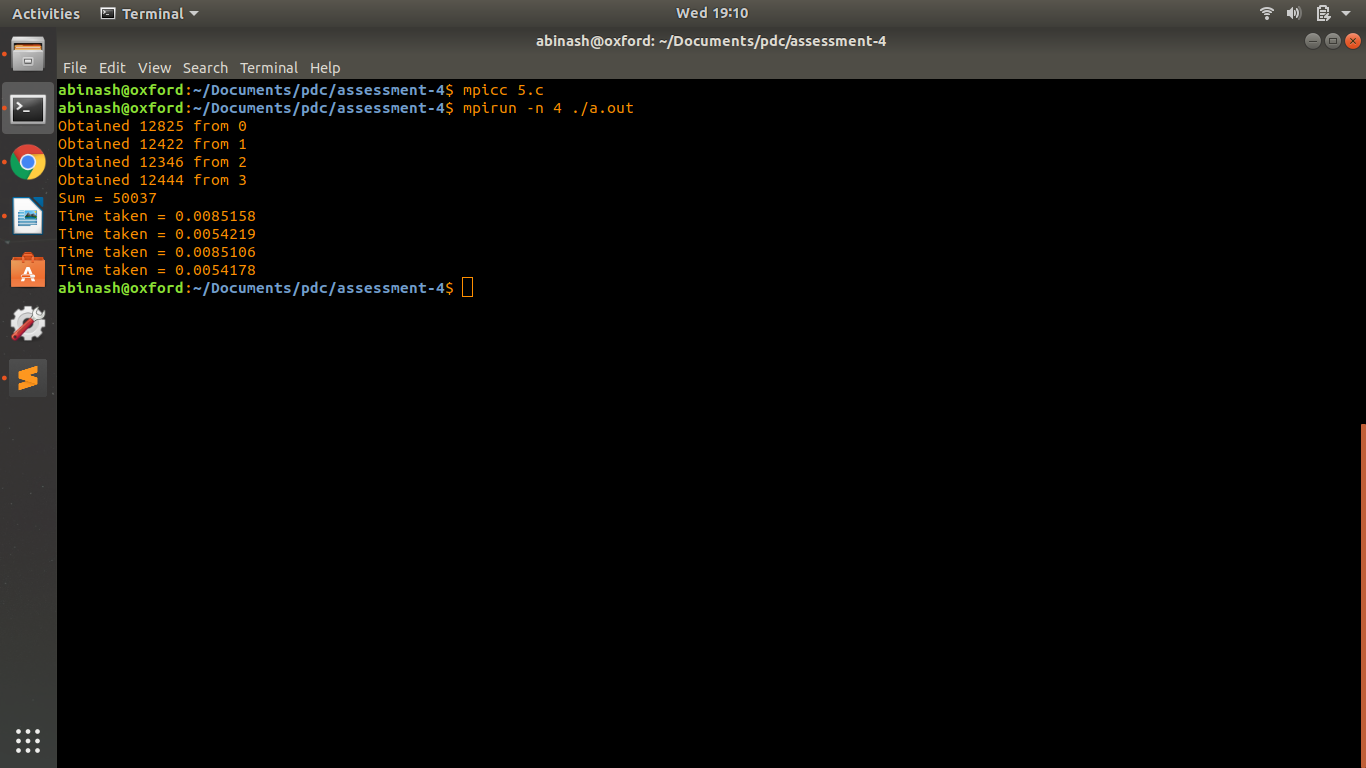
t2 = MPI\_Wtime();

MPI\_Finalize();

printf("Time taken = %.7f\n", (t2-t1));

return 0;

}



1. **Write an MPI program to calculate the the value of pi using broadcast and reduce functions.**

#include <stdio.h>

#include <mpi.h>

#include <math.h>

int main(int argc, char \*argv[]){

int done = 0, n, myid, numprocs, i, rc;

double PI25DT = 3.141592653589793238462643;

double mypi, pi, h, sum, x, a;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

while(!done){

if(myid==0){

printf("Enter the number of intervals: (0 quits)\n");

scanf("%d", &n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if(n==0)

break;

h = 1.0/(double)n;

sum = 0.0;

for(i=myid+1;i<=n;i=i+numprocs){

x = h\*((double)i - 0.5);

sum = sum + (4.0/(1.0+x\*x));

}

mypi = h \* sum;

MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if(myid==0){

printf("pi is approximately = %.16f\n", pi);

printf("Error = %.16f\n", fabs(pi-PI25DT));

}

MPI\_Finalize();

}

return 0;

}

