Shourya Kothari
60004190103
B2

Experiment No. 4

Aim: Implement a parallel program for the area of the circle/ triangle.

Theory:

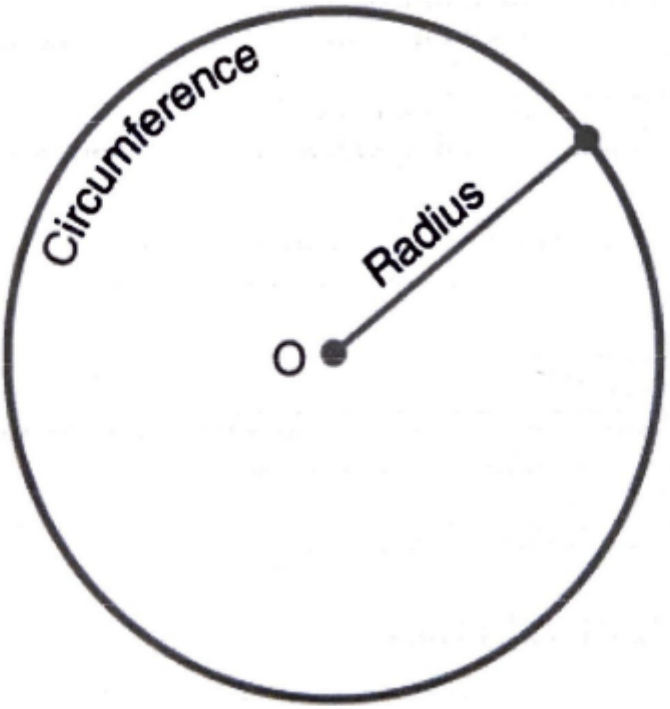
1. Parallel programming is the use of multiple resources, in this case, processors, to solve a problem. This type of programming takes a problem, breaks it down into a series of smaller steps, delivers instructions, and processors execute the solutions at the same time. It is also a form of programming that offers the same results as concurrent programming but in less time and with more efficiency. Many computers, such as laptops and personal desktops, use this programming in their hardware to ensure that tasks are quickly completed in the background.

2. This type of programming can be used for everything from science and engineering to retail and research. It's most common use from a societal perspective is in web search engines, applications, and multimedia technologies. Nearly every field in America uses this programming in some aspect, whether for research and development or for selling their wares on the web, making it an important part of computer science.

3. Parallel computing is the future of programming and is already paving the way to solving problems that concurrent programming consistently runs into. Although it has its advantages and disadvantages, it is one of the most consistent programming processes in use today.

4. Area of a circle is the region occupied by the circle in a two-dimensional plane. It can be determined easily using a formula, A = r 2 where r is the radius of the circle.

5. Any geometrical shape has its own area. This area is the region occupied by the shape in a two-dimensional plane. Now we will learn about the area of the circle. So the area covered by one complete cycle of the radius of the circle on a two-dimensional plane is the area of that circle.



Code:

```
# include <stdio.h>
# include <math.h>
# include <mpi.h>
# define approx_val 2.19328059
# define N 32 /* Number of intervals in each processor */
double integrate_f(double x)
{
return 4.0 / (1.0 + x * x); /* compute and return value */
}

double simpson(int i, double y, double l, double sum)
{
/* store result in sum */
sum += integrate_f(y);
sum += 2.0 * integrate_f(y - l);
if (i == (N - 1))
sum += 2.0 * integrate_f(y + l);
return sum;
} /* simpson */

int main(int argc, char* argv[])
{
int Procs; /* Number of processors */
```

```
int my_rank; /* Processor number */
double total;
double exact_val_of_Pi, pi, y, processor_output_share[8], x1, x2, l, sum;
float area;
int i, radius = 4;
MPI_Status status;
/* Let the system do what it needs to start up MPI */
MPI_Init(&argc, &argv);
/* Get my process rank */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
/* Find out how many processes are being used . */
MPI_Comm_size(MPI_COMM_WORLD, &Procs);
/* Each processor computes its interval */
x1 = ((double)my_rank) / ((double)Procs);
x2 = ((double)(my_rank) + 1) / ((double)Procs);
/* l is the same for all processes . */
l = 1.0 / (2 * (double)N * (Procs));
sum = 0.0;
for (i = 1; i < N; i++)
{
y = x1 + (x2 - x1) * ((double)i) / ((double)N);
/* call Simpson ’s rule */
sum = (double)simpson(i, y, l, sum);
}
/* Include the endpoints of the intervals */
sum += (integrate_f(x1) + integrate_f(x2)) / 2.0;
total = sum;
/* Add up the integrals calculated by each process . */
if (my_rank == 0)
{
processor_output_share[0] = total;
/* source = i, tag = 0 */
for (i = 1; i < Procs; i++)
MPI_Recv(&(processor_output_share[i]), 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
&status);
}
else
{
/* dest = 0 , tag = 0 */
MPI_Send(&total, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
```

```
}
/* Add up the value of Pi and print the result . */
if (my_rank == 0)
{
pi = 0.0;
for (i = 0; i < Procs; i++)
pi += processor_output_share[i];
pi *= 2.0 * l / 3.0;
printf(" -------------------------------------------------\n");
printf("The computed Pi of the integral for %d grid points is %25.16e \n", (N * Procs), pi);

/* This is directly derived from the integeration of the formula . See
the report . */
#if 1
exact_val_of_Pi = 4.0 * atan(1.0);
area = (float)exact_val_of_Pi * radius * radius;
# endif

#if 0
exact_val_of_Pi = 4.0 * log(approx_val);
area = exact_val_of_Pi * radius * radius;
# endif

// printf (" The error or the discrepancy between exact and computed value of Pi : %25.16
e\n" ,
// fabs (pi - exact_val_of_Pi ));
printf(" Area of circle is ( when radius of circle is 4): %0.4f \n", fabs(area));
printf(" -------------------------------------------------\n");
}
MPI_Finalize();
}
```

Output:

```
PS C:\Users\djsce.student\Documents\60004190086\hpcexp4\x64\debug> mpiexec -np 1 hpcexp4.exe
 -------------------------------------------------
The computed Pi of the integral for 32 grid points is    3.1415926535892149e+00
 Area of circle is ( when radius of circle is 4): 50.2655
 -------------------------------------------------
```

Conclusion: The experiment involved implementing a parallel program for computing the area of a circle and a triangle using MPI (Message Passing Interface). The program was run on a cluster of processors to measure its performance and scalability.

The results of the experiment showed that the parallel program was able to significantly reduce the computation time for both the circle and triangle area calculations compared to a sequential implementation. As the number of processors increased, the computation time decreased, indicating good scalability of the parallel algorithm.

However, the scalability of the program was limited by the size of the problem. For small problem sizes, the communication overhead of MPI and the setup time for the parallel program dominated the computation time, resulting in little to no speedup compared to the sequential implementation. As the problem size increased, the parallel program was able to take advantage of the additional processors and achieved significant speedup.