1. INTRODUCTION

1.1 AIM

To Implement Monte Carlo Methods with Parallelization using OpenMP and MPI which includes pseudo random number generators with GSL, Monte Carlo Markov Chain and implementation of practical applications like Monte Carlo integration, determination of temperature on 2D plate and simulation of neutron transport.

1.2 OBJECTIVES

- Learn about Monte Carlo methods.
- Study Monte Carlo Markov Chains.
- Implement Parallel methods on practical applications of Monte Carlo Methods.
- Experiment with the generation of random numbers using PRNG.
- Understand and apply the GNU Scientific Library.

1.3 MODULES

1.3.1 Module – 1

- Temperature inside a 2D plate
- Solving Integration problem using monte carlo approximation
- Parallel pseudo random number generation

1.3.2 Module – 2

- PRNG Using GNU Scientific Library
- Parallel program to implement Monte Carlo Markov Chain

1.3.3 Module – 3

- Parallel Monte Carlo tree search

1.3.4 Module – 4

- Parallel event based Monte Carlo program for a 1D region slab
- Stratified Sampling
- Deterministic Simpson's Rule

2. PROJECT OVERVIEW

A Monte Carlo method is an algorithm which solves a problem through the use of statistical sampling. The name is derived from the resort city in Monaco, famous for its games of chance. While early work in this field began in the nineteenth century, the first important use of Monte Carlo methods came into picture during the atomic bomb development during World War I

The Monte Carlo method is the only practical way to evaluate integrals of arbitrary functions in six or more dimensions. It has many other uses, including predicting the future level of the Dow Jones Industrial Average, solving partial differential equations, sharpening satellite images, modelling cell populations, and finding approximate solutions to NP-hard problems in polynomial time.

MODULES

2.1 Module – 1

- Temperature inside a 2D plate
- Solving Integration problem using monte carlo approximation
- Parallel pseudo random number generation

2.2 Module – 2

- PRNG Using GNU Scientific Library
- Parallel program to implement Monte Carlo Markov Chain

2.3 Module – 3

- Parallel Monte Carlo tree search

2.4 Module – 4

- Parallel event based Monte Carlo program for a 1D region slab
- Stratified Sampling
- Deterministic Simpson's Rule

3. MODULE DESCRIPTION

3.1 MODULE - 1

3.1.1 TEMPERATURE INSIDE A 2D PLATE

Imagine a very thin plate of homogeneous material. We wish to compute the steady-state temperature at a particular point in the plate. The top and the bottom of the plate are insulated, and the temperature at any point is solely determined by the temperatures surrounding it, except for the temperatures at the edges of the plate, which are fixed.

In this case the temperature at a point is the average of the temperatures of the points above it, below it, to its right. and to its left (which we can think of as "north,";"south;' "east," and "west").We can use a Monte Carlo technique to find the temperature at a particular point *S.* We compute the temperature of *S* by randomly choosing one of the four neighbours and adding ilS temperature to an accumulator. After we have sampled a random neighbour's temperature *n* times, we divide the sum by *n* to yield the temperature of *S.*

3.1.2 MONTE CARLO INTEGRATION

Monte Carlo Integration is a technique for numerical integration using random numbers. It is a particular Monte Carlo method that numerically computes a definite integral. Monte Carlo randomly chooses points at which the integrand is evaluated. This method is particularly useful for higher dimensional integrals. There are different methods to perform a Monte Carlo Integration, such as uniform sampling, stratified sampling and importance sampling. We applied the method of uniform sampling by taking uniform samples in the given limits.

3.1.3 PARALLEL PSEUDO RANDOM NUMBER GENERATOR (PPRNG)

The linear congruential method is more than 50 years old, and it is still the most popular. Linear congruential generators produce a sequence *X,* of random integers using this formula where *a* is called the multiplier, *c* is called the additive constant, and *M* is called the modulus. In some implementations *c* = O. When c: = 0, it is called a multiplicative Congruential generator. All three values must be carefully chosen in order to ensure that the sequence has a long period and good randomness properties. The maximum period is *M.*

3.2 MODULE 2

3.2.1 PRNG USING GNU SCIENTIFIC LIBRARY (GSL)

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License. The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite. Random Functions available in GSL are used and random numbers are generated.

3.2.2 PARALLEL PROGRAM TO IMPLEMENT MONTE CARLO MARKOV CHAIN

For computations involving high-dimensional posterior distributions, it is usually impractical to implement pure Monte Carlo solutions. Instead, the most effective strategy is often to construct a Markov chain with equilibrium distribution equal to the posterior distribution of interest. There are a variety of strategies that can be used to achieve this. Consider generation of a standard normal random quantity using a random walk Metropolis–Hastings sampler with $U(-\alpha, \alpha)$ innovations (α is a pre–chosen fixed "tuning" parameter).

3.3 MODULE -3

3.3.1 PARALLEL MONTE CARLO TREE SEARCH

Monte Carlo Tree Search (MCTS) methods are a relatively new (established in 2006) application of Monte Carlo methods and have seen much use in the field of AI for games (both classic board games and modern computer games). Essentially, this method is used as an alternative to a binary (or n-ary) tree search when the construction and storage of such a tree would be computationally prohibitive.

MCTS concentrates on analysing the most promising moves, basing the expansion of the search tree on random sampling of the search space. MCTS in games is based on many playouts. In each playout, the games are played-out to the very end by selecting moves at random. The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.

Each round of MCTS consists of four steps

- *Selection*: starting from root, select successive child nodes down to a leaf node. The section below says more about a way of choosing child nodes that lets the game tree expand towards most promising moves, which is the essence of MCTS.
- *Expansion*: unless leaf node ends the game, either create one or more child nodes of it and choose from them.
- *Simulation*: play a random playout from selected node.
- *Back propagation*: using the result of the playout, update information in the nodes on the path from selected node to root.

3.4 MODULE – 4

3.4.1 PARALLEL MONTE CARLO PROGRAM FOR A 1D REGION SLAB

We consider a simplified model of neutron transport in two dimensions. A source emits neutrons against a homogeneous plate having thickness *H* and infinite height. A neutron may be reflected by the plate, absorbed by the plate, or it may pass through the plate. We wish to compute the frequency at which each of these events occurs as a function of plate thickness *H.* Two constants that describe the interaction of the neutrons in the plate are the cross section of the capture *Cc* and the cross section of the scattering c,. The total cross section C = C, + C,.

The distance L a neutron travels in the plate before interacting with an atom is modelled by an exponential distribution with mean *l/C.* if *u* is a random number from the unifonn dis1ribution [0, I), the formula

L = --(1/C)*lnu

is *a* random number from the appropriate exponential probability density function. When a neutron interacts with an atom in the plate, thc probability of bouncing off the atom is *C.* While the probability of being absorbed by the atom is C. We may use a random number from the uniform distribution [.0, I) to determine the outcome of a neutron-atom interaction. If a neutron scatters, it has an equal probability of moving in any direction. Hence its new direction *D* (measured in radians) can be modelled by a random variable uniformly distributed between 0 and 1r. (Since the plate is infinite height, we do not need to distinguish between bouncing upward and bouncing downward.) Given direction *D,* the actual distance in the *x* direction the neutron travels in the plate between collisions is *L* cos *D.*

The simulation of a neutron continues until one of the following events occurs:

1. The neutron is absorbed by an atom.
2. The *x* position of the neutron is less than 0, meaning the neutron has been reflected by the plate.
3. The *x* position of the neutron is greater than *H,* meaning the neutron has been transmitted through the plate.

3.4.2 STRATIFIED SAMPLING METHOD

In statistics, stratified sampling is a method of sampling from a population. In statistical surveys, when subpopulations within an overall population vary, it is advantageous to sample each subpopulation (stratum) independently. Stratification is the process of dividing members of the population into homogeneous subgroups before sampling. The strata should be mutually exclusive: every element in the population must be assigned to only one stratum. The strata should also be collectively exhaustive: no population element can be excluded. Then simple random sampling or systematic sampling is applied within each stratum. This often improves the representativeness of the sample by reducing sampling error. It can produce a weighted mean that has less variability than the arithmetic mean of a simple random sample of the population.

In computational statistics, stratified sampling is a method of variance reduction when Monte Carlo methods are used to estimate population statistics from a known population.

3.4.3 DETERMINISTIC SIMPSON'S RULE

In numerical analysis, **Simpson's rule** is a method for numerical integration, the numerical approximation of definite integrals, The method is credited to the mathematician Thomas Simpson (1710–1761) of Leicestershire, England. Kepler used similar formulas over 100 years prior. Initially a points will be selected in the graph and substituted in the function and the value is obtained the points are substituted in the Simpsons rule and the average is obtained.

4. CODE LISTING

4. 1 Source Code explanation

MODULE - 1

TEMPERATURE INSIDE A 2D PLATE

```
/* File-name: monte_temp_detection.c
 * Purpose : To determine the temperature on a 2D plate which is being
heated from outside
 * Compile : cc monte_temp_detection.c -lpthread -lm
 * Run     : ./a.out
 * Inputs  :1.Select the size of the square matrix nxn
 *          2.Enter the surrounding temperature in degrees
 *          3.Select the coordinates on the plate where the temperature
need to determined
 * Outputs :1.Temperature will be displayed
 *          2.Graph associated with samples will be displayed X Denotes the
given input coordinates
 *                                                                  . Denotes the
path of random walk
 *                                                                  * Denotes the
point where the thread came into contact where the temperature is known
 */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<pthread.h>

int count[4];
double a[20][20]; //elements of the graph
char display[20][20]; //stores the path of random walk
double x,y; //coordinates and the copy of the coordinates
int org_x,org_y,sample;
double temp,req_temp;
int i,j; //loop counters
int m,n; //rows and columns of the graph
void visited(int mk_z_x,int mk_z_y);
void *Rand_walk(void* rank);
void Rand_walk_display();

void main()
{
 long thread;
 pthread_t* thread_handles;


 printf("Enter the Size of the plane : ");
 scanf("%d %d",&m,&n);

 for(i=0;i<m;i++)
    for(j=0;j<n;j++)
      display[i][j]=' ';

 printf("\n2D PLANE IS INITIALIZED\n");
 for(i=0;i<m;i++)
 {
   for(j=0;j<n;j++)
```

```
   {
     a[i][j]=0;
     printf("%.2lf ",a[i][j]);
   }
   printf("\n");
 }
 srandom(0);                              //seeding random number generator
 printf("\nEnter the temperature arround the plane     : ");
 scanf("%lf",&temp);

 printf("\nSET PROBABILITIES\n");
 for(i=0;i<m;i++)
 {
   for(j=0;j<n;j++)
   {
     a[i][j]= rand()%100;               //initializing the plane with
random numbers
     a[i][j]=a[i][j]*0.01;
     printf("%.2lf ",a[i][j]);
   }
   printf("\n");
 }
 printf("\nEnter the Coordinates from (0,0) to (%d,%d) : ",m-1,n-1);
 scanf("%lf%lf",&x,&y);
 org_x = (int)x;
 org_y = (int)y;  //storing the copies of original coordinates

 thread_handles=malloc(4*sizeof(pthread_t));
//allocating memory for threads
 for(i=0;i<4;i++)
   pthread_create(&thread_handles[i],NULL,Rand_walk,(void*)i);

 for(i=0;i<4;i++)
   pthread_join(thread_handles[i],NULL);
 free(thread_handles);

 req_temp=0;
 Rand_walk_display();


 for(i=0;i<4;i++)                    //temperature calculation
  req_temp=temp*(count[i]%4)+req_temp;
 req_temp=req_temp/4;
 printf("\nEstimated Temperature at the point (%d,%d) is
%.2lf\n",org_x,org_y,req_temp);
 printf("\nSamples Considered :
%d\n",sample=count[0]+count[1]+count[2]+count[3]);
 printf("\nEfficiency of the process : %.2lf",(sample/(m*n)));

}//end of main

/* Purpose : Random walk is performed in the initialized graph depending on
the assigned probabilities
 * Input   : Thread Rank
 * Output  : Random Walk on the graph will be obtained
 */
void *Rand_walk(void* rank)
{
 double north,south,east,west,selected_dir;  //directions of the points
 char* my_dir;
 int my_x,my_y;
```

```
int my_count=0;
int my_rank=(int)rank;
if(my_rank==0)
{
  my_dir="north";
  north=a[my_x=(int)x-1][my_y=(int)y];
}
else if(my_rank==1)
{
  my_dir="west";
  west=a[my_x=(int)x][my_y=(int)y+1];
}
else if(my_rank==2)
{
  my_dir="south";
  south=a[my_x=(int)x+1][my_y=(int)y];
}
else if(my_rank==3)
{
  my_dir="east";
  east=a[my_x=(int)x][my_y=(int)y-1];
}

while(1)
{
 west=a[my_x][my_y+1];    //west
 east=a[my_x][my_y-1];    //east
 north=a[my_x-1][my_y];    //north
 south=a[my_x+1][my_y];    //south

 if((north>south)&&(north>east)&&(north>west))
 {
   selected_dir=north;
   display[my_x][my_y]='.';
   my_x=my_x-1;
 }
 else if((south>east)&&(south>west))
 {
   selected_dir=south;
   display[my_x][my_y]='.';
   my_x=my_x+1;
 }
 else if((east>west))
 {
   selected_dir=east;
   display[my_x][my_y]='.';
   my_y=my_y-1;
 }
 else
 {
   selected_dir=west;
   display[my_x][my_y]='.';
   my_y=my_y+1;
 }
 my_count++;
 visited(my_x,my_y);
 if(my_x==0||my_y==0||my_x==20||my_y==20)
 {
   display[my_x][my_y]='*';
   break;
 }
```

```
 }
 count[my_rank]=my_count;
  return NULL;
}//end of Rand_walk function

/* Purpose : To make the visited node zero so as to avoid redundancy
 * Input   : Visited node on the graph
 * Output  : Visited nodes on the grpahs will be marked zero
 */
void visited(int mk_z_x,int mk_z_y)
{
  a[mk_z_x][mk_z_y]=0;
}
/* Purpose : To display the graph with the defined symbols initially
 * Input   : No inputs required
 * Output  : A graph displaying the visited nodes and originally selected
co ordinates etc.,
 */
void Rand_walk_display()
{
  printf("\nRandom Walk\n");
  display[org_x][org_y]='X';
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      printf("%c ",display[i][j]);
    }
    printf("\n");
  }

}
```

MONTE CARLO INTEGRATION

```
/* File Name : monte_carlo_integration.c
 * Purpose   : To apply Monte Carlo Integration on a Function and compare
with actual result
 * Process   : Conside f(x) as a function whose limits are inbetween [a,b]
 *             we are going to generate the random numbers x1,x2,x3.....xn
in between the interval [a,b] and
 *             estimate the average and compare with the result obtained
from actual integration of the function
 *             we took simple function f(x) = x^2+3*x+3 to process
 * Compile   : cc monte_carlo_integration.c -fopenmp
 * Run       : ./a.out
 * Inputs    : 1.For convenience we took limits from 0 to 1 and random
numbers will be generated in between the limits
 *             2.Given number of samples to be considered
 * Outputs   : 1.Gives the result of monte carlo integration is applied
 *             2.Actual result when original integration is applied
 */
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define PI 3.1416

void monte_func(double a,double b,int n);
double count[20000];
```

```
double sum=0;

void main()
{
 double I;   //x-is function variable y-function output
 int i;        //n-number of samples i- loop counter
 int threads;
 I = (1/3+3/2+3);    //substituting 1 in the integral of f(x)
 printf("\nValue obtained after integrating mathematically       :
%lf\n",I);
 printf("\nEnter Number of threads                                 : ");
 scanf("%d",&threads);
 #pragma omp parallel num_threads(threads)
 {
   monte_func(0,1,threads);
 }

 for(i=0;i<threads;i++)
   sum=sum+count[i];                    //Summation of substitutions
   sum=sum/threads;                     //Average with total number of
samples
 printf("\nIntegral Value obtained by applying Monte Carlo Method :
%lf\n",sum);
}

/* Purpose : To generate random numbers and substitute in the function and
store the value
 * Inputs  : Lower and upper limit and total number of samples
 * Output  : An Array with results of random number substitutions
 */

void monte_func(double a,double b,int n)
{
 float x;
 int thread_num=omp_get_thread_num();
 if(x= (random()%100)*0.01)
 {
   printf("\n %d thread executed with random number %.2f \n",thread_num,x);
   count[thread_num]= x*x+3*x+3;
   printf("\nCount[%d]  : %lf\n",thread_num,count[thread_num]);
 }
}
```

PARALLEL PSEUDO RANDOM NUMBER GENERATOR (PPRNG)

```
/* File Name : mc_prng.c
 * Purpose   : To develop a Pseudo Random Number Generator using linear
congruential method
 * Process   : Initially we seed a random number generator and threads
generate the random numbers
 *             using initial input Additive Constant and multipliers  in
the given period
 * Compile   : cc mc_prng.c -fopenmp
 * Execute   : ./a.out
 * Inputs    : Seed Value,Maximum Period,Random numbers required
 * Outputs   : List of random numbers
 */

#include<stdio.h>
#include<omp.h>
```

```
int x0;        //seed value
int a=2;         //multiplier
int c=1;         //Additive constant
int M;         //Maximum Period
int xi;        //next value
int no;        //number of random numbers
int i;         //loop counter
int acc[100];  //Accumulator to store numbers
void calc_rand_no(int x0,int M,int n0);
void main()
{
 int my_rank;
 int num_threads;
 int my_seed;
 printf("\nEnter the Seed value : ");
 scanf("%d",&x0);
 printf("\nEnter Maximum Period : ");
 scanf("%d",&M);
 printf("\nEnter the number of random numbers required : ");
 scanf("%d",&no);
 #pragma omp parallel num_threads(4) private(xi)
 {
   my_rank= omp_get_thread_num();
   my_seed=x0*my_rank;
   xi=my_seed;
   #pragma omp for
   for(i=0;i<no;i++)
    {
     xi= (xi*a+c)%M;
     printf("\nRandom Number generated by thread %d is %d\n",my_rank,xi);
    }
 }
}//end main

/* Purpose : To generate the random numbers
 * Input   : seed value,period ans initial value
 * Output  : Random numbers
 */
void calc_rand_no(int x0,int M,int no)
{
  int my_rank= omp_get_thread_num();
  int num_threads= omp_get_num_threads();
  int my_seed=x0*my_rank;
  printf("\nSeed Value of %d is %d\n",my_rank,my_seed);
  #pragma omp critical
  {
    for(i=1;i<no;i++)
    {
     acc[i]= (acc[i-1]*a+c)%M;
     printf("\nRandom Number generated by thread %d is
%d\n",my_rank,acc[i]);
    }
  }


}
```

MODULE 2

PRNG USING GNU SCIENTIFIC LIBRARY (GSL)

```
/* File Name : monte_prng_gsl.c
 * Purpose   : To implement PRNG using GSL(GNU Scientific Library)
 * Compile   : gcc monte_prng_gsl.c `pkg-config --cflags --libs gsl` -o
monte_prng_gsl -fopenmp
 * Execute   : ./monte_prng_gsl
 *             GSL Library need to be installed before executing
 */

#include<stdio.h>
#include<omp.h>
#include <gsl/gsl_rng.h>

int x0;        //seed value
int a=2;       //multiplier
int c=1;       //Additive constant
int M;         //Maximum Period
int xi;        //next value
int no;        //number of random numbers
int i;         //loop counter
int acc[100];  //Accumulator to store numbers
void calc_rand_no(int x0,int M,int n0);
void main()
{
 const gsl_rng_type * T;
 gsl_rng * r;
 int my_rank;
 int num_threads;
 int my_seed;
 gsl_rng_env_setup();

 T = gsl_rng_default;
 r = gsl_rng_alloc (T);
 x0 = gsl_rng_uniform (r)*100;
 printf("\nSeed value taken using GSL is : %d",x0);
 printf("\nEnter Maximum Period : ");
 scanf("%d",&M);
 printf("\nEnter the number of random numbers required : ");
 scanf("%d",&no);
 #pragma omp parallel num_threads(4) private(xi)
 {
   my_rank= omp_get_thread_num();
   my_seed=x0*my_rank;
   xi=my_seed;
   #pragma omp for
   for(i=0;i<no;i++)
    {
     xi= (xi*a+c)%M;
     printf("\nRandom Number generated by thread %d is %d\n",my_rank,xi);
    }
 }

 gsl_rng_free (r);
}//end main


void calc_rand_no(int x0,int M,int no)
```

```
{
  int my_rank= omp_get_thread_num();
  int num_threads= omp_get_num_threads();
  int my_seed=x0*my_rank;
  printf("\nSeed Value of %d is %d\n",my_rank,my_seed);
  #pragma omp critical
  {
    for(i=1;i<no;i++)
    {
     acc[i]= (acc[i-1]*a+c)%M;
     printf("\nRandom Number generated by thread %d is
%d\n",my_rank,acc[i]);
    }
  }
}
```

PARALLEL PROGRAM TO IMPLEMENT MONTE CARLO MARKOV CHAIN

```
/* File Name : mcmc.c
 * Purpose   : To Implement Markov Chain Monte Carlo using GSL
 * Process   : Using GSL we will seed the random numbers and those will be
written in a file
 *             And we used MPI to generate threads and to communicate among
them
 * Compile   : mpicc mcmc.c `pkg-config --cflags --libs gsl` -o mcmc
 * Run       : mpiexec mcmc (no. of numbers) (Any String)
 * Outputs   : A File with random numbers
 */


#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <mpi.h>
int main(int argc,char *argv[])
{
 const gsl_rng_type * T;
 int k,i,iters,ierr,my_rank; double x,can,a,alpha; gsl_rng *r;
 FILE *s; char filename[15];
 T = gsl_rng_default;
 MPI_Init(&argc,&argv);
 MPI_Comm_rank(MPI_COMM_WORLD,&k);
 if ((argc != 3))
 {
 if (k == 0)
 printf("Usage: %s <iters> <alpha>\n",argv[0]);
 MPI_Finalize();
 return(EXIT_FAILURE);
 }

 iters=atoi(argv[1]);
 alpha=atof(argv[2]);

 r=gsl_rng_alloc(T);

 sprintf(filename,"chain-%03d.tab",k);
 s=fopen(filename,"w");
 if (s==NULL)
 {
  perror("Failed open");
```

```
  MPI_Finalize();
  return(EXIT_FAILURE);
 }
 x = gsl_ran_flat(r,-20,20);
 fprintf(s,"Iter X\n");
 for (i=0;i<iters;i++)
 {
 can = x + gsl_ran_flat(r,-alpha,alpha);
 a = gsl_ran_ugaussian_pdf(can) / gsl_ran_ugaussian_pdf(x);
 if (gsl_rng_uniform(r) < a)
 x = can*my_rank;
 fprintf(s,"%d Thread printed %f",k,x);
 fprintf(s,"%d %f\n",i,x);
 }
 fclose(s);
 MPI_Finalize();
 return(EXIT_SUCCESS);
}
```

4.3 MODULE -3

PARALLEL MONTE CARLO TREE SEARCH

```
public class ElapsedTimer {
    // allows for easy reporting of elapsed time
    long oldTime;

    public ElapsedTimer() {
        oldTime = System.currentTimeMillis();
    }

    public long elapsed() {
        return System.currentTimeMillis() - oldTime;
    }

    public void reset() {
        oldTime = System.currentTimeMillis();
    }

    public String toString() {
        // now resets the timer...
        String ret = elapsed() + " ms elapsed";
        reset();
        return ret;
    }

    public static void main(String[] args) {
        ElapsedTimer t = new ElapsedTimer();
        System.out.println("ms elasped: " + t.elapsed());
    }
}

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class JEasyFrame extends JFrame {
    public Component comp;
    public JEasyFrame(Component comp, String title) {
        super(title);
        this.comp = comp;
```

```
        getContentPane().add(BorderLayout.CENTER, comp);
        pack();
        this.setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        repaint();
    }
}
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

public class TreeNode {
    static Random r = new Random();
    static int nActions = 5;
    static double epsilon = 1e-6;

    TreeNode[] children;
    double nVisits, totValue;

    public void selectAction() {
        List<TreeNode> visited = new LinkedList<TreeNode>();
        TreeNode cur = this;
        visited.add(this);
        while (!cur.isLeaf()) {
            cur = cur.select();
            // System.out.println("Adding: " + cur);
            visited.add(cur);
        }
        cur.expand();
        TreeNode newNode = cur.select();
        visited.add(newNode);
        double value = rollOut(newNode);
        for (TreeNode node : visited) {
            // would need extra logic for n-player game
            // System.out.println(node);
            node.updateStats(value);
        }
    }

    public void expand() {
        children = new TreeNode[nActions];
        for (int i=0; i<nActions; i++) {
            children[i] = new TreeNode();
        }
    }

    private TreeNode select() {
        TreeNode selected = null;
        double bestValue = Double.MIN_VALUE;
        for (TreeNode c : children) {
            double uctValue =
                    c.totValue / (c.nVisits + epsilon) +
                            Math.sqrt(Math.log(nVisits+1) / (c.nVisits +
epsilon)) +
                            r.nextDouble() * epsilon;
            // small random number to break ties randomly in unexpanded
nodes
            // System.out.println("UCT value = " + uctValue);
            if (uctValue > bestValue) {
                selected = c;
                bestValue = uctValue;
```

```
            }
        }
        // System.out.println("Returning: " + selected);
        return selected;
    }

    public boolean isLeaf() {
        return children == null;
    }

    public double rollOut(TreeNode tn) {
        // ultimately a roll out will end in some value
        // assume for now that it ends in a win or a loss
        // and just return this at random
        return r.nextInt(2);
    }

    public void updateStats(double value) {
        nVisits++;
        totValue += value;
    }

    public int arity() {
        return children == null ? 0 : children.length;
    }
}
import java.util.LinkedList;
import java.util.List;
public class TreeNodeTest {
    public static void main(String[] args) {
        List<TreeNode> list = new LinkedList<TreeNode>();
        list.add(null);
        System.out.println("list: " + list);
        TreeNode tn = new TreeNode();
        ElapsedTimer t = new ElapsedTimer();
        int n = 1000;
        for (int i=0; i<n; i++) {
            tn.selectAction();
        }
        System.out.println(t);
        TreeView tv = new TreeView(tn);
        tv.showTree("After " + n + " play outs");
        System.out.println("Done");
    }
}
import javax.swing.*;
import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.HashMap;

/* Simple TreeView for a MCTS Tree
 */

public class TreeView extends JComponent {
    TreeNode root;
    int nw = 30;
    int nh = 20;
    int inset = 20;
    int minWidth = 300;
    int heightPerLevel = 40;
    Color fg = Color.black;
```

```
    Color bg = Color.cyan;
    Color nodeBg = Color.white;
    Color highlighted = Color.red;
    // the highlighted set of nodes...
    HashMap<TreeNode, Color> high;

    public TreeView(TreeNode root) {
        this.root = root;
        high = new HashMap<TreeNode, Color>();
    }

    public void paintComponent(Graphics gg) {
        // Font font =
        // g.setFont();
        Graphics2D g = (Graphics2D) gg;
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        int y = inset;
        int x = getWidth() / 2;
        g.setColor(bg);
        g.fillRect(0, 0, getWidth(), getHeight());
        draw(g, root, x, y, (int) (1.1 * getWidth()) - inset * 0);
    }

    private void draw(Graphics2D g, TreeNode cur, int x, int y, int wFac) {
        // draw this one, then it's children

        int arity = cur.arity();
        for (int i = 0; i < arity; i++) {
            if (cur.children[i].nVisits > 0) {
                int xx = (int) ((i + 1.0) * wFac / (arity + 1) + (x - wFac
/ 2));
                int yy = y + heightPerLevel;
                g.setColor(fg);
                g.drawLine(x, y, xx, yy);
                draw(g, cur.children[i], xx, yy, wFac / arity);
            }
        }
        drawNode(g, cur, x, y);
    }

    private void drawNode(Graphics2D g, TreeNode node, int x, int y) {
        String s = (int) node.totValue + "/" + (int) node.nVisits;
        g.setColor(nodeBg);
        // if (high.contains(node)) g.setColor(highlighted);
        g.fillOval(x - nw / 2, y - nh / 2, nw, nh);
        g.setColor(fg);
        g.drawOval(x - nw / 2, y - nh / 2, nw, nh);
        g.setColor(fg);
        FontMetrics fm = g.getFontMetrics();
        Rectangle2D rect = fm.getStringBounds(s, g);
        g.drawString(s, x - (int) (rect.getWidth() / 2), (int) (y +
rect.getHeight() / 2));
    }

    public Dimension getPreferredSize() {
        // should make this depend on the tree ...
        return new Dimension(minWidth, heightPerLevel * (10 - 1) + inset *
2);
    }
```

```
    public TreeView showTree(String title) {
        new JEasyFrame(this, title);
        return this;
    }
}
```

MODULE – 4

PARALLEL MONTE CARLO PROGRAM FOR A 1D REGION SLAB

```
/* File Name : monte_neutron_trans.c
 * Purpose   : To Implement an Event based Monte Carlo Program on a 1D
Region
 * Process   : 1.take input for number of neutrons
 *             2.Calculate Scattering and Absorbing Component
 *             3.Perform required arthmetic
 * Compile   : cc monte_neutron_trans.c -fopenmp
 * Run       : ./a.out
 */
#include<stdio.h>
#include<math.h>
#include<omp.h>

#define PI 3.14
void monte_neutron(double C_c,double C_s,double C);
int r=0,b=0,t=0; //counts of reflected,absorbed,transmitted neutrons
int a=1; //true while particle still bouncing
int n; //Number of samples
double x=0; //position of particle (0<=x<H)
double u; //random number
double d = 180; //Direction of neutron
double L; //Distance Neutron Travelled before collision
int H=150; //Thickness of a plate
double C_c; //Absorbing Component
double C_s; //Scattering Component
double C; //mean distance

void main()
{
 int i; //loop counter

 C_c = random()%100;
 C_s = random()%100;
 C = C_c+C_s;
 printf("\nEnter number of neutrons : ");
 scanf("%d",&n);
 #pragma omp parallelnum_threads(n)
 monte_neutron(C_c,C_s,C);

 printf("\nNeutrons Reflected,Absorbed and Transmitted are :
%d,%d,%d\n",r,b,t);
}
```

```
/* Purpose : To determine if neutron is scattered, reflected or transmitted
 * Inputs  : Scattered and Absorbed Component
 * Outputs : No. of neutrons scattered,absorbed and transmitted
 */
void monte_neutron(double C_c,double C_s,double C)
{
  int i;
  for(i=0;i<n;i++)
 {
   d=0;
   x=0;
   a=1;
   u=(random()%100)*0.01;
   while(a)
   {
    L = -(1/C)*log(u);
    x = x+L*cos((d*PI)/180);
    if(x<0) //Reflected
    {
     r=r+1;
     printf("\n*---->--*             |");
     printf("\n          *           |");
     printf("\n*--*-<--*             |");
     printf("\n                      |");
     a=0;
    }
    else if(x>=H) //Transmitted
    {
     t=t+1;
     a=0;
     printf("\n*--->---*---->-----*--|->-*");
     printf("\n                      |");
    }
    else if(u<(C_c/C)) //Absorbed
    {
     b=b+1;
     a=0;
     printf("\n*---->--*             |");
     printf("\n          *           |");
     printf("\n   *-<--*             |");
     printf("\n                      |");
    }
    else
      d=u*3.14;
   }//end while
  }//end for
}
```

STRATIFIED SAMPLING METHOD

```
/* File Name : mc_stratified_sampling.c
 * Purpose   : To Implement Stratified Sampling using monte carlo method
 * Process   : Take number of stratums and take their size also
 *             Take the total number of samples required
 *             Collect indivdual samples from stratums using the below
formula
 *             Sample_stratum =
(sample_stratum_size)*(req_sample/total_sample_size)
 *             Paradigm - employee selection for a project
 * Compile   : cc mc_stratified_sampling.c -fopenmp
 * Run       : ./a.out
 */

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

int monte_stratified(int sample_stratum_size,int req_sample,int
total_samples_size);
int count[5];
void main()
{
  int mft; //male full time employees
  int mpt; //male part time employees
  int fft; //female full  time employees
  int fpt; //female part time employees
  int project_size;  //No of Employees required for the project
  int emp_size; //Total Number of Employees available
  int pmft; //No. of full time working men selected for project
  int pmpt; //No. of part time working men selected for project
  int pfft; //No. of full time working women selected for project
  int pfpt; //No. of part time working women selected for project
  int thread_num;

  printf("\nEnter No. of male full time working employees   : ");
  scanf("%d",&mft);
  printf("\nEnter No. of male part time working employees   : ");
  scanf("%d",&mpt);
  printf("\nEnter No. of female full time working employees : ");
  scanf("%d",&fft);
  printf("\nEnter No. of female part time working employees : ");
  scanf("%d",&fpt);
  emp_size=mft+mpt+fft+fpt;
  printf("\nTotal No. of Employees in the organization      :
%d\n",emp_size);
  printf("\nEnter the No. of employees required for project ; ");
  scanf("%d",&project_size);
  #pragma omp parallel num_threads(4)
  {
   thread_num = omp_get_thread_num();
     if(thread_num==0)
      pmft=monte_stratified(mft,project_size,emp_size);
       else if(thread_num==1)
        pmpt=monte_stratified(mpt,project_size,emp_size);
       else if(thread_num==2)
        pfft =monte_stratified(fft,project_size,emp_size);
       else
         pfpt = monte_stratified(fpt,project_size,emp_size);
```

```
}
}
/* Purpose : To calculate No. of Employees required from each stratums
 * Input   : sample size of individual stratum, required sample and total
number of samples
 * Output  : No. of male and female full and part time employees
 */

int monte_stratified(int sample_stratum_size,int req_sample,int
total_samples_size)
{
  int sample_stratum=0,thread_num;
  float temp;
  thread_num = omp_get_thread_num();
  sample_stratum = (sample_stratum_size*req_sample)/total_samples_size;
  if(thread_num==0)
  printf("\nNo. of male full time working employees     :
%d\n",sample_stratum);
  else if(thread_num==1)
  printf("\nNo. of female full time working employees   :
%d\n",sample_stratum);
  else if(thread_num==2)
  printf("\nNo. of male part time working employees     :
%d\n",sample_stratum);
  else
  printf("\nNo. of female part time working employees   :
%d\n",sample_stratum);
}
```

DETERMINISTIC SIMPSON'S RULE

```
/* File Name : monte_simpsons.c
 * Purpose   : To apply simpsons rule and determine the area of the
function
 * Compile   : cc monte_simpsons.c -fopenmp
 * Run       : ./a.out
 */



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

double count_x[50],count_fx[50];
double f(double x);


void main()
{
 double a,b,n; //lower and upper bound and total number of samples required
 double h; //period or
 int i,j;
 double temp,sum=0;
 printf("Enter the Lower and Upper Bounds : ");
 scanf("%lf%lf",&a,&b);
 printf("\nTotal Number of Samples : ");
 scanf("%lf",&n);
```

```
 h=(b-a)/n;
 for(i=0;i<n;i++)
 {
  count_x[i]=random()%(int)(b-a+1)+a;
 }

 #pragma omp parallel num_threads((int)n)
 {
  count_fx[omp_get_thread_num()]=f(count_x[omp_get_thread_num()]);
 }
  sum=count_fx[0];
  for(j=1;j<n;j++)
  {
   if(j%2==0)
   count_fx[j]=2*count_fx[j];
   else
   count_fx[j]=4*count_fx[j];
   sum=sum+count_fx[j];
  }
  sum = (h/3)*sum;
 printf("Area of the curve according to Simpsons Rule is : %.2f\n",sum);

}//end of main

/* Purpose : To calculate the square of the number in x
 * Input   : value x
 * Output  : value x^2
 */
double f(double x) {
double return_val;
return_val = x*x;
return return_val;
} /* f */
```

4.2 Methods used in the programs

```
int MPI Init( int argc p                    /_ in/out _/,
              char  argv p                   /_ in/out _/);
```

The arguments, argc p and argv p, are pointers to the arguments to main, argc, and argv. However, when our program doesn't use these arguments, we can just pass NULL for both.

```
int MPI Finalize(void);
```

MPI Finalize tells the MPI system that we're done using MPI, and that any resources allocated for MPI can be freed.

```
int MPI Comm size(
                    MPI Comm comm /_ in _/,
                    int_ comm sz p /_ out _/ );
```

In MPI a **communicator** is a collection of processes that can send messages to each other. One of the purposes of MPI Init is to define a communicator that consists of all of the

processes started by the user when she started the program. This communicator is called MPI COMM WORLD.

```
int MPI Comm rank(
                  MPI Comm comm /_ in _/,
                  int_ my rank p /_ out _/);
```

MPI Comm rank returns in its second argument the calling process' rank in the communicator. We'll often use the variable comm sz for the number of processes in MPI COMM WORLD, and the variable my rank for the process rank.

```
int MPI Send(
void_ msg_buf p         /_ in _/,
int msg_size            /_ in _/,
MPI_Datatype msg_type   /_ in _/,
int dest                /_ in _/,
int tag                 /_ in _/,
MPI_Comm communicator   /_ in _/);
```

The first three arguments, msg_buf p, msg_size, and msg_type, determine the contents of the message. The remaining arguments, dest, tag, and communicator, determine the destination of the message. The first argument, msg_buf p, is a pointer to the block of memory containing the contents of the message. In our program, this is just the string containing the message, greeting. (Remember that in C an array, such as a string, is a pointer.) The second and third arguments, msg_size and msg_type, determine the amount of data to be sent. In our program, the msg size argument is the number of characters in the message plus one character for the '/0' character..

```
int MPI Recv(
void_ msg_buf p          /_ out _/,
int buf_size             /_ in _/,
MPI Datatype buf_type    /_ in _/,
int source               /_ in _/,
int tag                  /_ in _/,
MPI_Comm communicator    /_ in _/,
MPI_Status status p      /_ out _/);
```

Thus, the first three arguments specify the memory available for receiving the message: msg buf p points to the block of memory, buf size determines the number of objects that can be stored in the block, and buf type indicates the type of the objects. The next three arguments identify the message. The source argument specifies the process from which the message should be received. The tag argument should match the tag argument of the message being

sent, and the communicator argument must match the communicator used by the sending process.

```
int pthread create(
pthread t_ thread p              /_ out _/,
const pthread attr t_ attr p     /_ in _/,
void_ (_start routine)(void_)    /_ in _/,
void_ arg p                      /_ in _/);
```

Function starts a new thread in the calling process. The new thread starts execution by invoking *start_routine*(); *arg* is passed as the sole argument of *start_routine*().

```
int pthread join(
pthread t thread        /_ in _/,
void__ ret val p        /_ out _/);
```

The *pthread_join*() function shall suspend execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join*() call with a non-NULL *value_ptr* argument

```
# pragma omp parallel num threads(thread count)
```

The pragma *omp parallel* is used to fork additional threads to carry out the work enclosed in the construct in parallel. The original thread will be denoted as *master thread* with thread ID 0.

5. EXPERIMENT SETUP

5.1 Execution

Message Passing Interface

Complile :

```
$ mpicc  -g  -Wall  -o mpi hello  mpi hello.c
```

Execute :

Many systems also support program startup with mpiexec:

```
$ mpiexec -n <number of processes> ./mpi hello
```

So to run the program with one process, we'd type

```
$ mpiexec -n 1 ./mpi hello
```

and to run the program with four processes, we'd type

```
$ mpiexec -n 4 ./mpi hello
```

Pthreads

Compile :

```
$ gcc -g  -Wall  -o pth hello pth hello.c -lpthread
```

Execute:

To run the program, we just type

```
$ ./pth hello <number of threads>
```

Open MP

Compile :

To compile this with gcc we need to include the -fopenmp

```
$ gcc  -g  -Wall -fopenmp  -o omp hello omp hello.c
```

Execute:

To run the program, we specify the number of threads on the command line. For example, we might run the program with four threads and type

```
$ ./omp hello 4
```

5.2 Output/Execution

5.2.1 MODULE - 1

TEMPERATURE INSIDE A 2D PLATE

```
Enter the Size of the plane : 20 20

2D PLANE IS INITIALIZED
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

Enter the temperature arround the plane     : 60

SET PROBABILITIES
0.83 0.86 0.77 0.15 0.93 0.35 0.86 0.92 0.49 0.21 0.62 0.27 0.90 0.59 0.63 0.26 0.40 0.26 0.72 0.36
0.11 0.68 0.67 0.29 0.82 0.30 0.62 0.23 0.67 0.35 0.29 0.02 0.22 0.58 0.69 0.67 0.93 0.56 0.11 0.42
0.29 0.73 0.21 0.19 0.84 0.37 0.98 0.24 0.15 0.70 0.13 0.26 0.91 0.80 0.56 0.73 0.62 0.70 0.96 0.81
0.05 0.25 0.84 0.27 0.36 0.05 0.46 0.29 0.13 0.57 0.24 0.95 0.82 0.45 0.14 0.67 0.34 0.64 0.43 0.50
0.87 0.08 0.76 0.78 0.88 0.84 0.03 0.51 0.54 0.99 0.32 0.60 0.76 0.68 0.39 0.12 0.26 0.86 0.94 0.39
0.95 0.70 0.34 0.78 0.67 0.01 0.97 0.02 0.17 0.92 0.52 0.56 0.01 0.80 0.86 0.41 0.65 0.89 0.44 0.19
0.40 0.29 0.31 0.17 0.97 0.71 0.81 0.75 0.09 0.27 0.67 0.56 0.97 0.53 0.86 0.65 0.06 0.83 0.19 0.24
0.28 0.71 0.32 0.29 0.03 0.19 0.70 0.68 0.08 0.15 0.40 0.49 0.96 0.23 0.18 0.45 0.46 0.51 0.21 0.55
0.79 0.88 0.64 0.28 0.41 0.50 0.93 0.00 0.34 0.64 0.24 0.14 0.87 0.56 0.43 0.91 0.27 0.65 0.59 0.36
0.32 0.51 0.37 0.28 0.75 0.07 0.74 0.21 0.58 0.95 0.29 0.37 0.35 0.93 0.18 0.28 0.43 0.11 0.28 0.29
0.76 0.04 0.43 0.63 0.13 0.38 0.06 0.40 0.04 0.18 0.28 0.88 0.69 0.17 0.17 0.96 0.24 0.43 0.70 0.83
0.90 0.99 0.72 0.25 0.44 0.90 0.05 0.39 0.54 0.86 0.69 0.82 0.42 0.64 0.97 0.07 0.55 0.04 0.48 0.11
0.22 0.28 0.99 0.43 0.46 0.68 0.40 0.22 0.11 0.10 0.05 0.01 0.61 0.30 0.78 0.05 0.20 0.36 0.44 0.26
0.22 0.65 0.08 0.16 0.82 0.58 0.24 0.37 0.62 0.24 0.00 0.36 0.52 0.99 0.79 0.50 0.68 0.71 0.73 0.31
0.81 0.30 0.33 0.94 0.60 0.63 0.99 0.81 0.99 0.96 0.59 0.73 0.13 0.68 0.90 0.95 0.26 0.66 0.84 0.40
0.90 0.84 0.76 0.42 0.36 0.07 0.45 0.56 0.79 0.18 0.87 0.12 0.48 0.72 0.59 0.09 0.36 0.10 0.42 0.87
0.06 0.01 0.13 0.72 0.21 0.55 0.19 0.99 0.21 0.04 0.39 0.11 0.40 0.67 0.05 0.28 0.27 0.50 0.84 0.58
0.20 0.24 0.22 0.69 0.96 0.81 0.30 0.84 0.92 0.72 0.72 0.50 0.25 0.85 0.22 0.99 0.40 0.42 0.98 0.13
0.98 0.90 0.24 0.90 0.09 0.81 0.19 0.36 0.32 0.55 0.94 0.04 0.79 0.69 0.73 0.76 0.50 0.55 0.60 0.42
0.79 0.84 0.93 0.05 0.21 0.67 0.04 0.13 0.61 0.54 0.26 0.59 0.44 0.02 0.02 0.06 0.84 0.21 0.42 0.68

Enter the Coordinates from (0,0) to (19,19) : 10 10

Random Walk
                *
                . .
                  .
                  .
      . . . .     .
      . .   .     . .
        . . .       . . .
          . .           .
        . . .   . .     . .
      . .   . . . . . . . .
    . .   . . . . . x . .
* . .     .   . . . . . . . .
          .                 .
        . .               . . . . . .
      . .                 . . . . . .
* . . .
                                .
                                . . .
                                    .
                                . . . .
*                               . . . .
```

```
Estimated Temperature at the point (10,10) is 30.00
Samples Considered : 98
```

MONTE CARLO INTEGRATION

```
hitesh@hitesh-PC:~/Desktop/pp_mod1$ ./a.out
Value obtained after integrating mathematically       : 4.000000
Enter Number of threads                               : 10000
Integral Value obtained by applying Monte Carlo Method : 4.790771
```

PARALLEL PSEUDO RANDOM NUMBER GENERATOR (PPRNG)

```
hitesh@hitesh-PC:~/Desktop/pp_mod1$ cc mc_prng.c -fopenmp
hitesh@hitesh-PC:~/Desktop/pp_mod1$ ./a.out
Enter the Seed value : 10
Enter Maximum Period : 400
Enter the number of random numbers required : 5
Random Number generated by thread 1 is 21
Random Number generated by thread 1 is 61
Random Number generated by thread 2 is 41
Random Number generated by thread 0 is 1
Random Number generated by thread 0 is 3
```

5.2.2 MODULE - 2

PRNG USING GNU SCIENTIFIC LIBRARY (GSL)

```
hitesh@hitesh-PC:~/Desktop/pp_mod2$ gcc monte_prng_gsl.c `pkg-config
--cflags --libs gsl` -o monte_prng_gsl -fopenmp
hitesh@hitesh-PC:~/Desktop/pp_mod2$ ./monte_prng_gsl
Seed value taken using GSL is : 99
Enter Maximum Period : 500
Enter the number of random numbers required : 6
Random Number generated by thread 3 is 1
Random Number generated by thread 1 is 3
Random Number generated by thread 2 is 397
Random Number generated by thread 1 is 199
Random Number generated by thread 1 is 399
Random Number generated by thread 3 is 95
```

PARALLEL PROGRAM TO IMPLEMENT MONTE CARLO MARKOV CHAIN

```
hitesh@hitesh-PC:~/Desktop/pp_mod2$ mpicc mcmc.c `pkg-config --
cflags --libs gsl` -o mcmc
hitesh@hitesh-PC:~/Desktop/pp_mod2$ mpiexec mcmc 10 hello
hitesh@hitesh-PC:~/Desktop/pp_mod2$ cat chain-000.tab
Iter X
0 19.989670
1 19.989670
```

```
2 19.989670
3 19.989670
4 19.989670
5 19.989670
6 19.989670
7 19.989670
8 19.989670
9 19.989670
```

5.2.3 MODULE -3

PARALLEL MONTE CARLO TREE SEARCH

```
hitesh@hitesh-PC:~/Desktop/pp_mod3$ cc 34.c -fopenmp
hitesh@hitesh-PC:~/Desktop/pp_mod3$ ./a.out 15
enter the number of games to be played
100
no.of games won/no.of games played: 17/34
no.of games won/no.of games played: 18/33
no.of games won/no.of games played: 13/33
```

5.2.4 MODULE – 4

PARALLEL MONTE CARLO PROGRAM FOR A 1D REGION SLAB

```
hitesh@hitesh-PC:~/Desktop/pp_mod4$ cc monte_neutron_trans.c -
fopenmp -lm
hitesh@hitesh-PC:~/Desktop/pp_mod4$ ./a.out

Enter number of neutrons : 5

*--->---*---->-----*--|->-*
                       |
*---->--*              |
          *            |
   *-<--*              |
                       |
*--->---*---->-----*--|->-*
                       |
*---->--*              |
          *            |
   *-<--*              |
                       |
*--->---*---->-----*--|->-*
                       |
Neutrons Reflected,Absorbed and Transmitted are : 0,2,3
```

STRATIFIED SAMPLING METHOD

```
hitesh@hitesh-PC:~/Desktop/pp_mod4$ cc mc_stratified_sampling.c -
fopenmp
hitesh@hitesh-PC:~/Desktop/pp_mod4$ ./a.out

Enter No. of male full time working employees   : 100
Enter No. of male part time working employees   : 50
Enter No. of female full time working employees : 90
Enter No. of female part time working employees : 40
-----------------------------------------------------
Total No. of Employees in the organization      : 280
-----------------------------------------------------
Enter the No. of employees required for project : 60
No. of male full time working employees     : 21
No. of female full time working employees   : 10
No. of male part time working employees     : 19
No. of female part time working employees   : 8
```

DETERMINISTIC SIMPSON'S RULE

```
hitesh@hitesh-PC:~/Desktop/pp_mod4$ cc monte_simpsons.c -fopenmp
hitesh@hitesh-PC:~/Desktop/pp_mod4$ ./a.out
Enter the Lower and Upper Bounds : 0 5
Total Number of Samples : 100
Area of the curve according to Simpsons Rule is : 211.15
```

6. PERFORMANCE EVALUATION

Sl.no	Program	Serial Time(in ms)	Parallel Time(in ms)
1.	Temperature Detection	10.45322	6.594995
2.	Pseudo Random Number Generation	0.07	0.01
3.	Monte Carlo Integration	1.463771	0.732511
4.	PRNG Using GSL	12.22566	10.119707
5.	Monte Carlo Markov Chain	0.699599	0.041839
6.	Monte Carlo Tree Search	3.44443	1.91396
7.	Neutron Particle Transport	4.55554	2.130586
8.	Stratified Sampling	24.44544	11.178107
9.	Deterministic Simpson's Rule	0.004884	0.000684

Table 7.1 – Illustrating the difference in time of execution for serial and parallel code.

7. REFERENCES

- Alireza Haghighat *Monte Carlo Methods for Particle Transport*, First Edition, CRC Press.
- Michael J, Quinn *Parallel Programming in C with MPI and OpenMP*, International Edition 2003,Mc Graw Hill Publication.
- James E. Gentle *Random Number Generation and Monte Carlo Methods*, 2nd Edition, Springer Publications.
- Erricos John Kontoghiorghes *Handbook of parallel computing and statistics*,2005,Chapmann/HALL CRC Publications.
- GNU Scientific Library manual referred in the URL - https://www.gnu.org/software/gsl/manual/html_node/