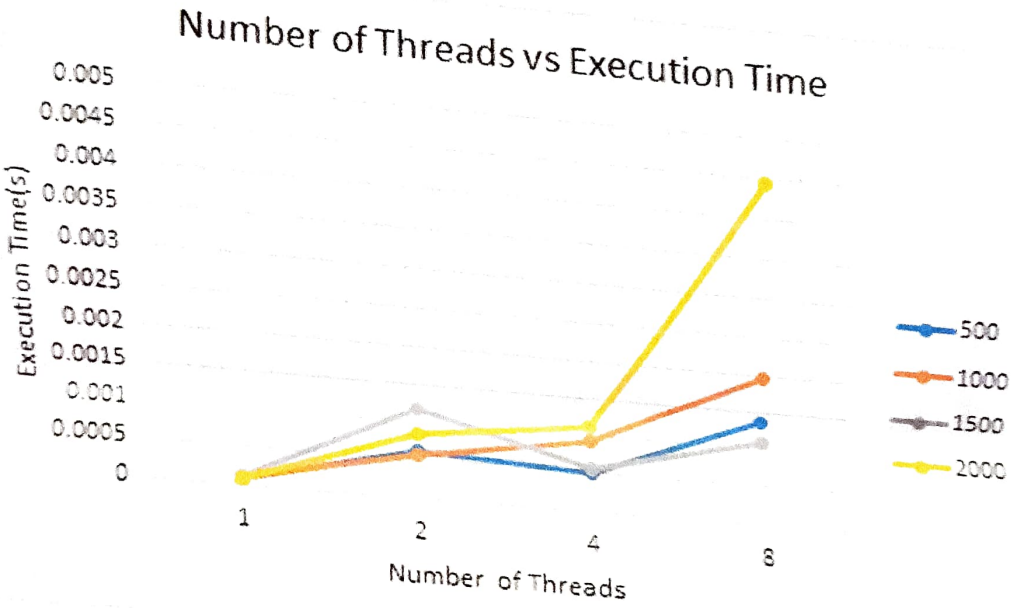
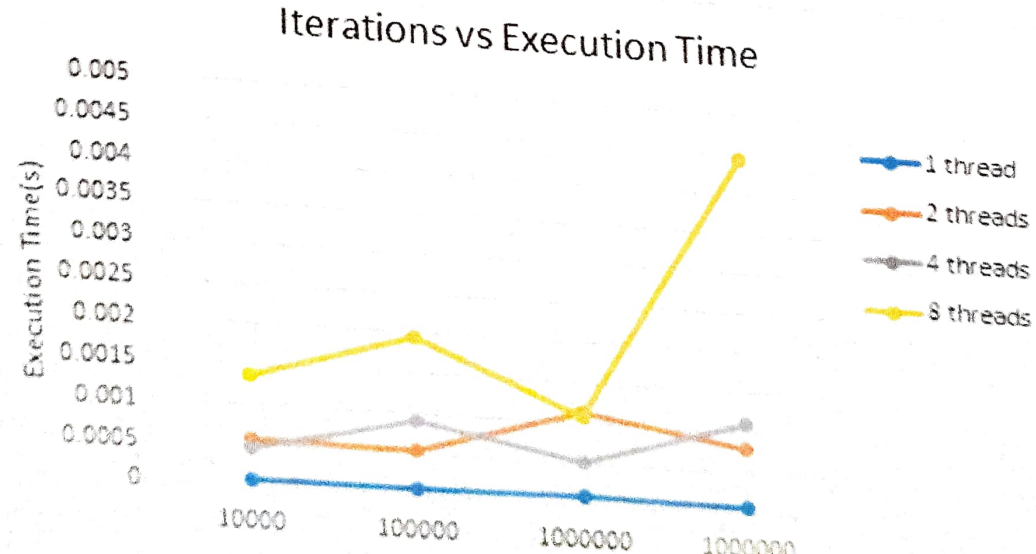
INDEX

INDEX

Iterations	Execution Time-	Number of	threads	
	1	2	4	8
10000	0.000068	0.000599	0.00048	0.001361
100000	0.000095	0.000549	0.000909	0.001967
1000000	0.000112	0.001156	0.000543	0.001086
1000000	0.000069	0.000819	0.001117	0.004463





Experiment No. 1A Name: ADITYA PATIL

DOMS Page No. 1
Date

AIM

To write an OpenMP program that computes value of PI using Monte Carlo algorithm.

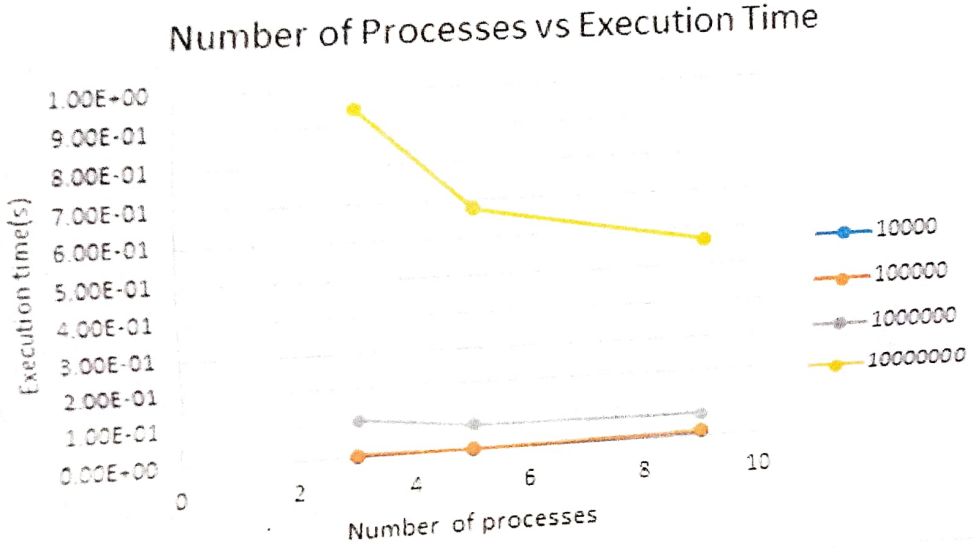
Program 1A

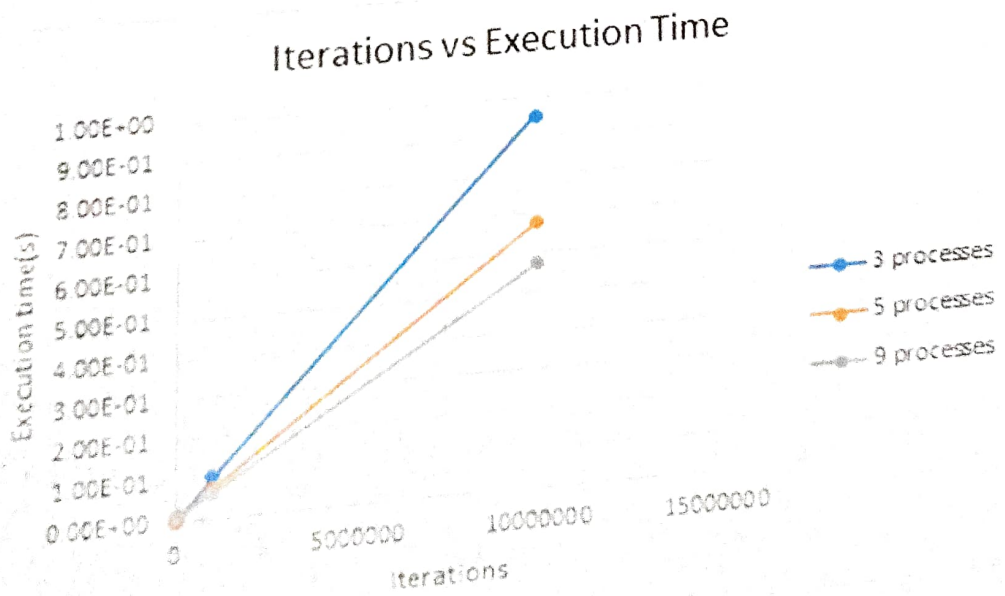
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define SEED 50123

int main()
{ long n=0, i, count=0;
  double x,y,z;
  srand (SEED);
  printf ("Size \t\t T1 \t\t T2 \t T4 \t T8");
  for( n=100; n<=1000000; n*=10)
    printf ("\n %ld \t", n);
    for(int t=1; t<=8; t*=2) {
      count=0;
      double start = omp_get_wtime();
#pragma omp parallel for private (x,y,z)
reduction (+: count) num_threads (t)
      for (i=0; i<n; i++)
      { x= (double) rand()/RAND_MAX;
```

Teacher's Signature:

MPI -MONTE CARLO

	Execution Time		
Iterations	3	5	9
10000	1.48E-03	8.57E-04	1.36E-02
100000	9.86E-03	7.58E-03	1.71E-02
1000000	0.105168	0.074548	0.065742
10000000	0.949605	0.67502	0.567609





Experiment No. 1

Name: ADITYA PATIL

```
            y = (double) rand() / RAND_MAX;
            z = x*x + y*y;
            if (z <= 1) count++;
        }
        double pi = (double) count / n * 4;
        double stop = omp_get_wtime();
        printf("%.1f %.1fs\t", pi, stop - start);
        }
    }
    printf("\n");
    return 0;
}
```

Program 1B

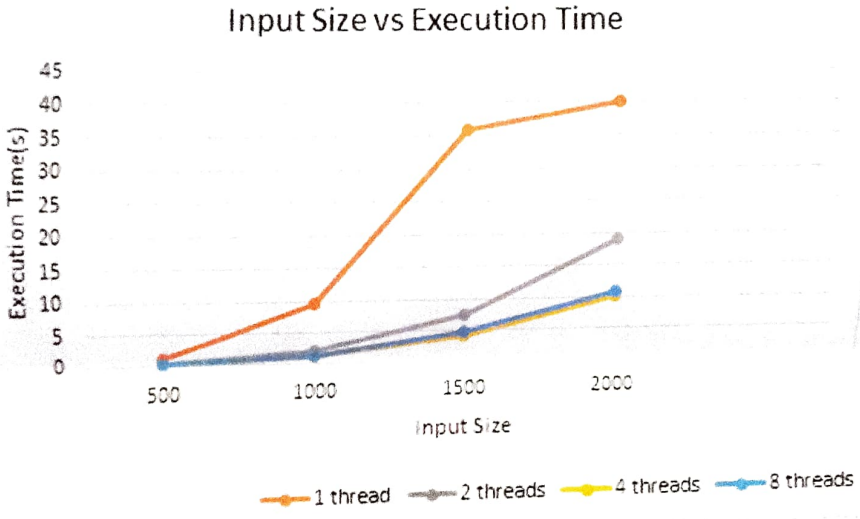
AIM

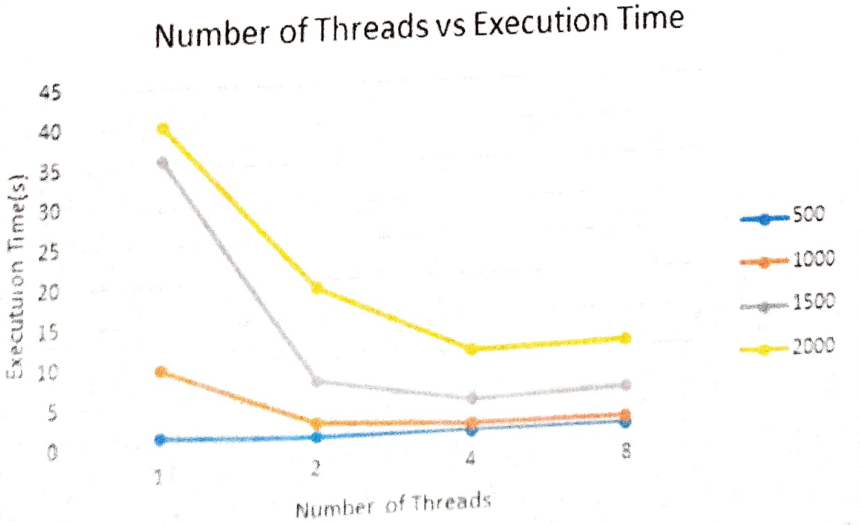
Write a MPI Program that computes the value of PI using Monte-Carlo Algorithm

Program 1B

```
#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
#define SEED 3655942
```

Input size	Execution Time (1 thread)	Execution Time (2 threads)	Execution Time (4 threads)	Execution Time (8 threads)
500	1.047931	0.219803	0.12051	0.126442
1000	9.559035	1.972103	1.072029	1.1002
1500	35.79007	7.371719	4.025677	4.717746
2000	40.05189	19.12589	10.25357	10.71961





Experiment No. 2 Name : ADITYA PATI

Date

Experiment No. 2 Name: ADITYA PATI

DOMS Page No. 9
Date

AIM

Write an OpenMP program that computes a simple matrix-matrix multiplication using dynamic memory allocation.

a) Illustrate the correctness of the program
b) Justify the inference when outer 'for loop' is parallelized with & without using the explicit data scope variables.

Program

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char * argv[]){
  printf("Size \t\t\t2\t4\t8");
  for(int n=500; n<=2000; n+=500){
    printf("\n%d\t", n);
    int **a, **b, **c;
    int i, j, k;
    a = (int **) malloc(n*sizeof(int*));
    b = (int **) malloc(n*sizeof(int*));
    c = (int **) malloc(n*sizeof(int*));
    for(i=0; i<n; i++){
      a[i] = (int*) malloc(n*sizeof(int));
```

Teacher's Signature:

```
b[i] = (int *) malloc (n * sizeof (int));
c[i] = (int *) malloc (n * sizeof (int));
for (i=0; i<n; i++) {
    a[i] * [j] = rand () % n;
    b[i][j] = rand () % n;
}}
for (int t=1; t<=8; t*=2) {
    double x = omp_get_wtime();
#pragma omp parallel for private (j,k) num_
threads (t)
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            c[i][j] = 0;
            for (k=0; k<n; k++){
                c[i][j] += a[i][k] * b[k][i]; }}
    double y = omp_get_wtime();
    printf ("%lf \t", y-x); }}
    printf ("\n");
    return 0;
}
```

Teacher's Signature:

Experiment No. 3 Name: ADITYA PATI

DOMS Page No. 6
Date

AIM

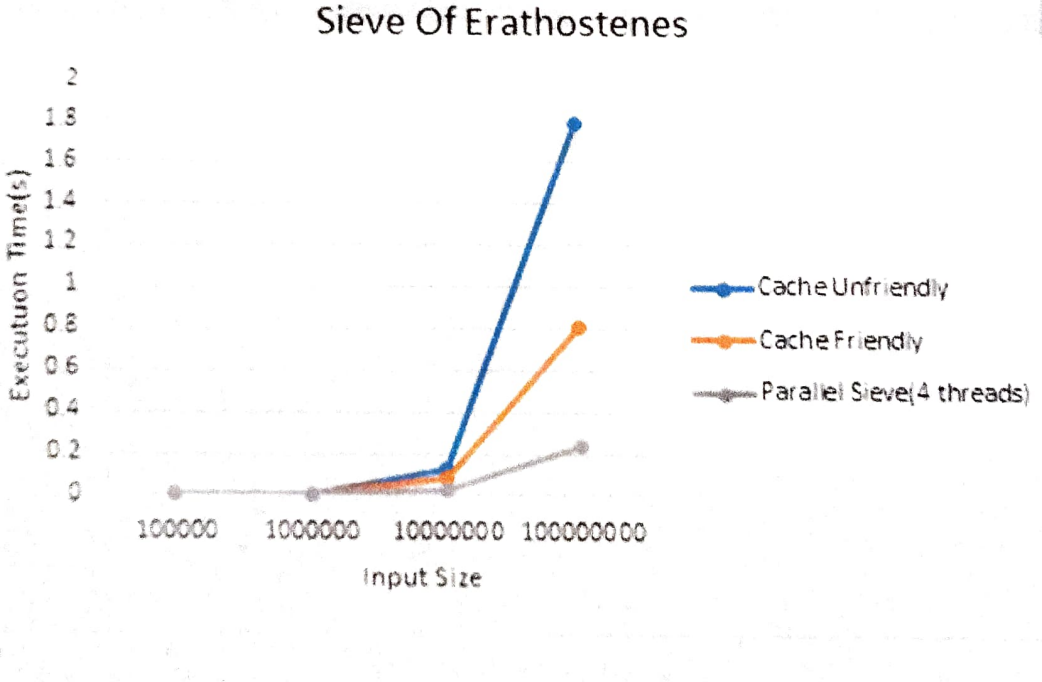
Write a program for Cache unfriendly Sieve of Eratosthenes, Cache friendly Sieve of Eratosthenes & Parallel Sieve for enumerating numbers upto N & prove the correctness.

Program

```
#include <math.h>
#include <string.h>
#include <omp.h>
#include <iostream>
using namespace std;
double t=0;

long Strike(bool Composite[], long i, long stride, long
    limit) { for (; i<=limit; i+=stride)
        Composite[i]= true;
    return ii}

long CacheUnfriendlySieve(long n){
    long count = 0, m=(long) sqrt((double) n);
    bool * composite = new bool[n+1];
    memset(composite, 0, n);
    t = omp_get_wtime();
```

Teacher's Signature:

Input size	Execution Time					
	Cache Unfriendly Sieve	Cache Friendly Sieve	Cache Parallel Sieve			
			1	2	4	8
100000	0.000818	0.000913	0.00103	0.001074	0.000896	0.001647
1000000	0.008269	0.009027	0.008996	0.005949	0.003532	0.004876
10000000	0.122772	0.079445	0.082135	0.052763	0.025498	0.019586
1E+08	1.79078	0.805944	0.797903	0.422097	0.245255	0.245255



```
for (long i=2; i<=m; i++)
  { if (!composite[i]) {
      ++count;
      Strike (composite, 2*i, i, n);
    }
  }

for (long i=m+1; i<=n; ++i)
    if (!composite[i]) ++count;
t = omp_get_wtime() - t;
delete[] composite;
return count; }

long ParallelSieve (long n) {
    long count=0; m=(long) sqrt((double)n),
    n_factor=0;
    long* factor = new long[m];
    t = omp_get_wtime();
#pragma omp parallel
  {
bool* composite = new bool[m+1]; long* striked
    =new long[m];
  #pragma omp single
  {
  memset(composite, 0, m);
  for (long i=2; i<=m; ++i)
```

Experiment No. 3 Name: ADITYA PATIL

DOMS Page No. 8
Date

```
if ( !composite, 2 * i, m)
    factor [n_factor++] = i;
}}
long base = -1;
#pragma omp for reduction (+:count)
for (long window = m+1; window<=n; window+=m)
{ memset (composite, 0, m);
    if (base != window) {
        base = window;
    }

int main() {
    long size = 10000, count;
printf ("Size \t\t Cache Unfriendly \t Cache
friendly \t Parallel Sieve \n");
    for (int i=1; i<=4; i++) { size = size*10;
printf ("%ld \t", size);
count = Cache Unfriendly Sieve (size);
printf ("%ld \t %f \t", count, t);
count = Parallel Sieve (size);
printf ("%ld \t %f \n", count, t);
}
return 0;
}
```

Teachers Signature:

Experiment No. 9 Name ADITYA PATIL

2015 Page No. 9
Date

AIM

Write a program to convert a colour image to black & white image.

a) Demonstrate the performance of different scheduling techniques for varying chunk values.

b) Analyze the scheduling patterns by assigning a single color value for an image for each thread

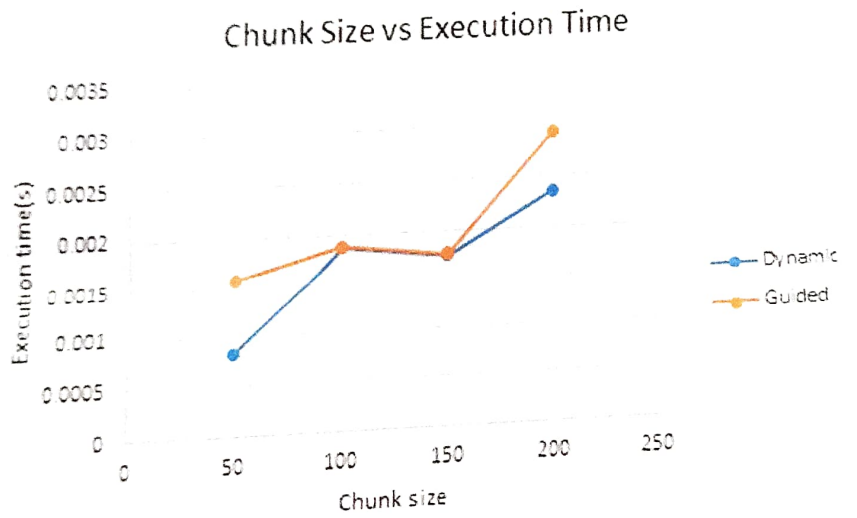
Program

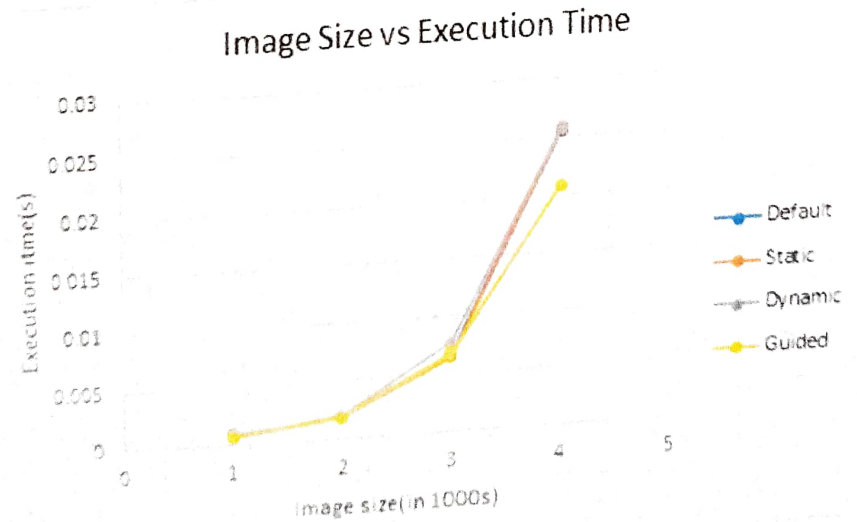
```
#include <stdio.h>
#include <errno.h>
#include <gd.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char **argv){
    FILE *fp, *fp1 = fopen();
    gdImagePtr img;
    char iname[16], oname[15];
    int color, x, y, i=0, red, green, blue, temp;
    long w, h;
    omp_sched_t def_sched; int def_chunk_size;
    omp_get_schedule(&def_sched, &def_chunk_size);
```

Teacher's Signature: _____

Scheduling Type			
Default	Static	Dynamic	Guided
0.00097	0.000713	0.000824	0.000786
0.001888	0.001904	0.002056	0.001899
0.00676	0.006501	0.007738	0.006864
0.025445	0.025573	0.025754	0.020828

Chunk Size	Scheduling Type: Dynamic	Guided
50	0.000824	0.001588
100	0.001875	0.001884
150	0.001751	0.001774
200	0.002351	0.002932





Experiment No. 4 Name: ADITYA PATI

DOMS Page No. 10
Date

```
for (int i=0; i<4; i++) {
    sprintf (iname, "in%d.png", i+1);
    for (int sched = 0x0; sched <= 0x3; sched++) {
        fp = fopen(iname, "r");
        sprintf (oname, "output%d%d.png", i+1, sched);
        img = gdImageCreateFromPng (fp);
        w = gdImageSX (img);
        h = gdImageSY (img);
        if (sched == 0x0)
            printf ("%ldx%ld\n", w, h);
        omp_set_schedule (def_sched, def_chunk_size);
    }
    else
        omp_set_schedule (sched, 0);
    double t = omp_get_wtime();
#pragma omp parallel for private (y, color, red,
    green, blue, tmp, tid)
        for (x=0; x<w; x++) {
            for (y=0; y<h; y++) {
                color = gdImageGetPixel (img, x, y);
                red = gdImageRed (img, color);
                green = gdImageGreen (img, colour);
                blue = gdImageBlue (img, color);
                red = green = blue = tmp;
                gdImageSetPixel (img, x, y, color); } }
```

Teacher's Signature:

Experiment No. 4 Name : ADITYA PATIL

DOMS Page No. 14
Date

```
t = omp_get_wtime() - t;
fp1 = fopen(oname, "w");
gdImagePng(img, fp1);
fclose(fp1)
gdImageDestroy(img);
printf("%.6f\t", t); }
printf("\n"); } return 0; }
```

Teacher's Signature: _____

Experiment No. 5 Name: ADITYA PATIL

DOMS Page No. 12
Date

AIM

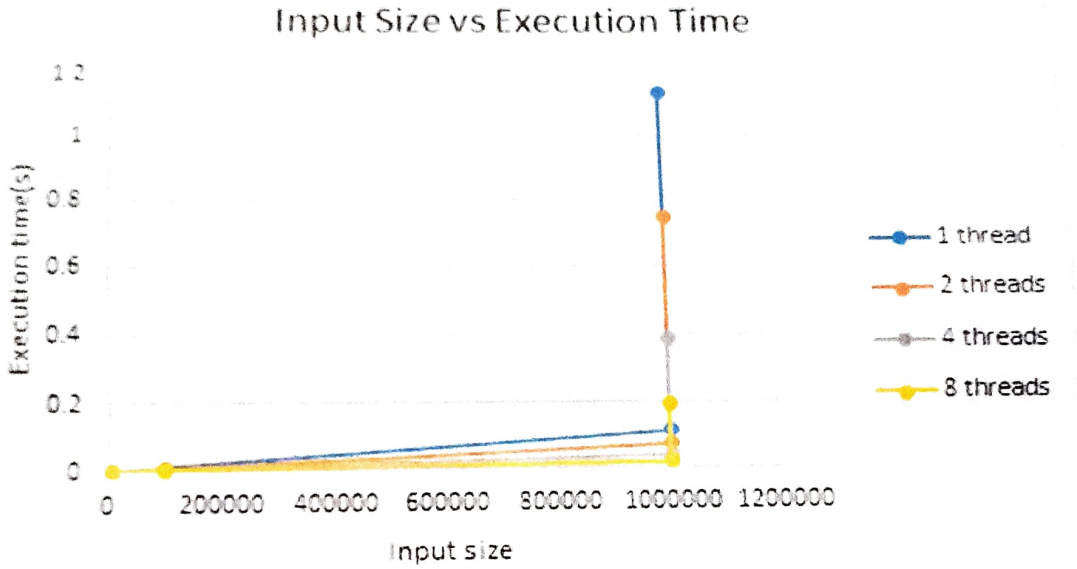
Write a parallel program for Points Classification. Justify the inference.

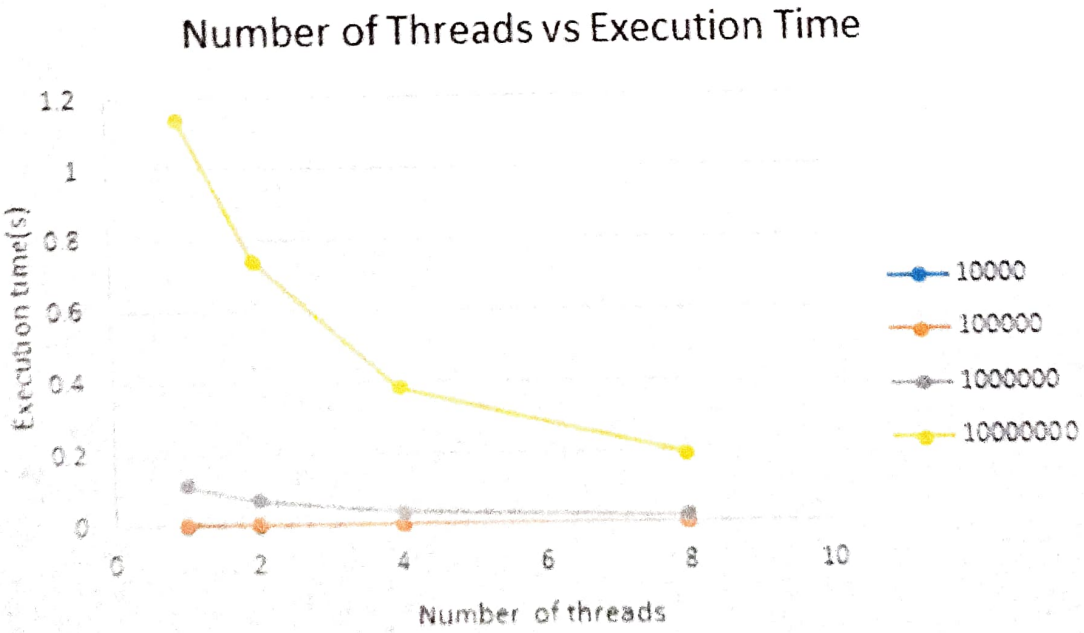
Program

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#define CLUSTER_SIZE 4
int cluster[CLUSTER_SIZE][2] = {{75,25},{25,25},{25,75},{75,75}};
long long cluster_count[CLUSTER_SIZE];
short points[10000000][2];
char output[10000] = "";
void populate_points(unsigned long long size){
    long long i;
    for(i=0; i<size; i++){
        srand(i);
        points[i][0] = rand()%100;
        points[i][1] = rand()%100;
    }}
double get_distance(int x1, int y1, int x2, int y2){
    int x = x2-x1, y = y2-y1;
```

Teacher's Signature

	Execution Time			
Input Size	1	2	4	8
10000	0.001208	0.001096	0.000922	0.00122
100000	0.011513	0.00773	0.00442	0.002891
1000000	0.114586	0.074563	0.039273	0.020269
1000000	1.141697	0.742433	0.383055	0.193872





```
    return (double) sqrt ((x*x) + (y*y));
}
int main(){
    double t;
    sprintf (output, "Size \t T1 \t T2 \t T8 \n");
for (int index =0; index<5; index++) {
    printf ("Size: %ld", points_sizes [index]);
    unsigned long long i;
    for (int nt=1; nt<9; nt*=2) {
        for (i=0; i< CLUSTER_SIZE; i++)
            cluster_count [i] = 0;
    t = omp_get_wtime();
# pragma omp parallel for reduction (+:cluster_count)
num_threads (nt)
    for (i=0; i< points_sizes [index] ; i++) {
    double min_dist =100, cur_dist =-1;
    int j, cluster_index =-1;
    for (j=0; j< CLUSTER_SIZE; j++) {
cur_dist = get_distance (points [i][0],
    points [i][1], cluster [j][0], cluster [j][1]);
        if (cur_dist < min_dist) {
            min_dist = cur_dist;
            cluster_index = j; }}
    puts (output);
    return 0; }
```

Teacher's Signature

Experiment No. 6 Name :

DOMS Page No. 14
Date

AIM

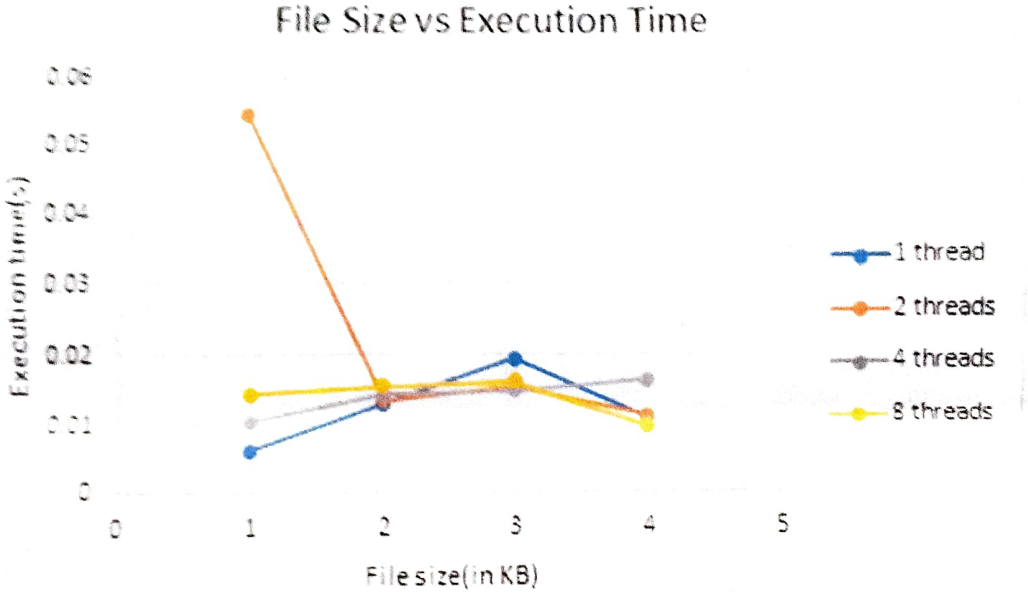
Write a parallel program for word search in a file. Justify the inference.

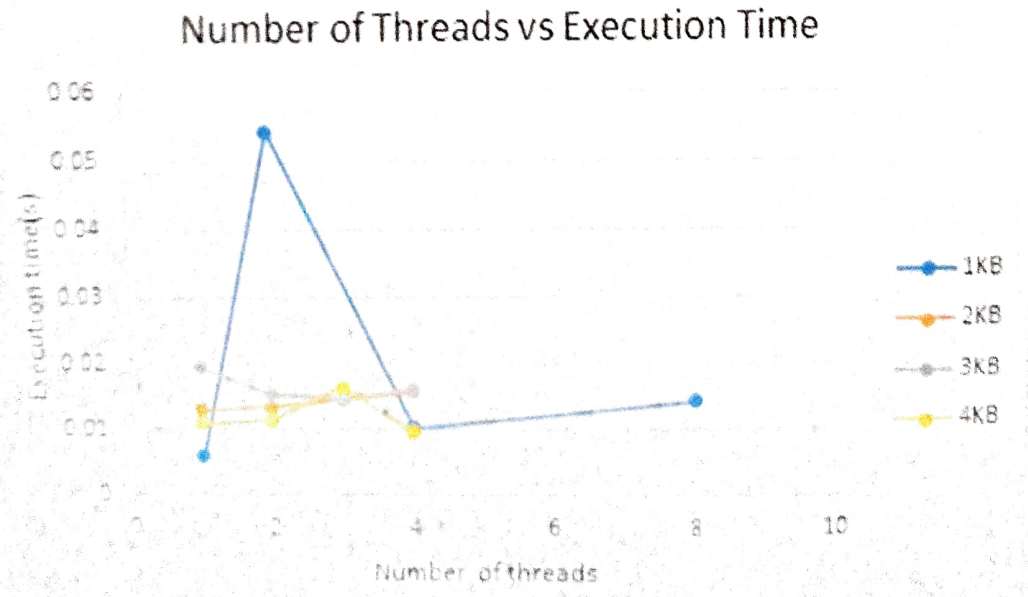
Program

```
#include <stdio.h>
#include <omp.h>
#include <ctype.h>
#include <string.h>
#define COUNT 10
char search_words[20][COUNT] = { "The", "around",
"graphics", "from", "by", "be", "a", "which", "various",
"mount" };
long counts[COUNT];
int is_equal( char *a, const char *key, int
    ignore_case) {
    int len_a = strlen(a), len_b = strlen(key);
    if (len_a != len_b) return 0;
    if (ignore_case) return strcasecmp(a, key) == 0;
    return strcmp(a, key) == 0;
}
void read_word(char *temp, FILE *fp) {
    int i=0; char ch;
```

Teacher's Signature:

	Execution Time			
File Size(KB)	1	2	4	8
1	0.006009	0.054372	0.010441	0.014479
2	0.013115	0.013621	0.014502	0.015667
3	0.019483	0.015392	0.014927	0.016296
4	0.01105	0.0115	0.016502	0.009686





```
while ((ch=fgetc(fp))!=EOF && isalpha(ch)
  ==0) {}
while (ch!=EOF && isalpha(ch)!=0) {
  temp[i++]=ch;
  ch = fgetc(fp); }
temp[i]='\0'; }

long determine_count (const char *file_name, const
  char *key, int ignore_cases) {
  int key_index=0, key_len=strlen(key);
  long word_count=0; char ch;
  FILE *fp = fopen(file_name, "r");
  char temp[50]; int i=0;
  while (feof(fp)==0) { read_word(temp,fp);
  if (is_equal(temp,key,ignore_case)!=0)
    word_count++; }
  return word_count; }
int main()
{ char output[1000]="";
  int i; for (i=0;i<COUNT;i++)
  counts[i]=0;
char *myfiles[4]={"file1.txt","file2.txt",
  "file3.txt","file4.txt");
  sprintf(output,"Size\tT1\tT2\tT4\tT5");
```

Teachers Signature :

```
for (int iter=0; iter<4; iter++){
  FILE *fp = fopen(img_folder[iter], "r");
  fseek(fp, 0L, SEEK_END);
  for (int t=1; t<=8; t*=2) {
    omp_set_num_threads(t);
    double start = omp_get_wtime();
#pragma omp parallel for
    for (int t=1; t<=8; t*=2){
      omp_set_num_threads(t);
      sprintf(output, "%s %lf %lf\t", output, time);
    }
  for (i=0; i<COUNT; i++)
    printf("%s: %ld", search_words[i],
           count[i]);
    }
printf("\n");
puts(output);
return 0;
}
```

Teacher's Signature _____

Experiment No. 7 Name :

DOMS Page No. 17
Date

AIM

Write a MPI-C program Multi-Tasking. Justify the inference.

Program

```
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void p0_set_input (int *input1, int *input2) {
    *input1 = 10000000;
    *input2 = 100000;
  printf ("\n P0_SET_PARAMETERS", *input1, *input2); }

void p0_send_input (int input1, int input2) {
  int id = 1, tag = 1;
  MPI_Send (&input1, 1, MPI_INT, id, tag, MPI_COMM_
        WORLD); id = 2; tag = 2;
  MPI_Send (&input2, 1, MPI_INT, id, tag, MPI_
        COMM_WORLD); }
```

Teacher's Signature:

```
aditya@aditya-Lenovo-ideapad-320-15IKB:~/Dow
10 January 2021 05:30:01 PM

MPI_MULTITASK:
  C / MPI version

P0_SET_PARAMETERS:
  Set INPUT1 = 10000000
      INPUT2 = 100000
  Process 2 time = 1.47705
  Process 1 time = 7.72909

  Process 1 returned OUTPUT1 = 615
  Process 2 returned OUTPUT2 = 9592
  Process 0 time = 7.72904

MPI_MULTITASK:
  Normal end of execution.
10 January 2021 05:30:08 PM
```

```
int p1_recieve_input() {
    int id=0, input1, tag=1; MPI_Status status;
MPI_Recv(&input1, 1, MPI_INT, id, tag, MPI_COMM_WORLD,
  &status); return input1; }

int p1_compute_output(int input1) {
  int i, j, k, output1=0;
    for(i=2; i<=output1; i++) {
    j=i; k=0;
    while(i<j) {
     if((j%2)==0) j=j/2;
    else j=3*j+1;
     k++;
   }
   if(output<k) output1=k; }
      return output;
  }
void p1_send_output(int output1) {
   int id=0, tag=3; MPI_Send(&output, 1, MPI_INT, id, tag,
     MPI_COMM_WORLD);
  }
int p2_recieve_input() {
    int id=0, input2, tag=2; MPI_Status status;
MPI_Recv(&input2, 1, MPI_INT, id, tag, MPI_COMM_
WORLD, &status); return input2; }
```

```
int p2_compute_output(int input2){
    int i, j, output2=0, prime;
    for(i=2; i<=input2; i++){
        prime = i;
        for(j=2; j<i; j++){
            if(i%j==0){
                prime=0; break;}}}
    return output2;}

void timestamp(){
#define TIME_SIZE 40
    static char time_buffer[TIME_SIZE];
    const struct tm *tm;
    time_t now;
    now = time(NULL);
    tm = localtime(&now);
    printf("%s\n", time_buffer);
#undef TIME_SIZE}

int main(int argc, char **argv){
    int id, ierr, input1, input2, output1, output2, p;
    double wtime;
    ierr = MPI_Init(&argc, argv);
    if(ierr!=0){
                exit(1);}
```

Experiment No. 7 Name :

DOMS Page No. 20
Date

```
ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &id);
MPI_Comm_size ( MPI_COMM_WORLD, &p);
if (p<3) {
    MPI_Finalize ();
    exit (1); }
} if (id == 0) {
    timestamp ();
wtime = MPI_Wtime (); p0_set_input (&input1, &input2);
p0_recieve_output (&output1, &output2);
wtime = MPI_Wtime () - wtime;
MPI_Finalize ();
timestamp (); }
else if (id == 1) {
    wtime = MPI_Wtime (); input1 = p1_recv_input();
output1 = p1_compute_output (input1);
p1_send_output (output1);
 MPI_Finalize ();
int id = 0, input1, tag = 1; MPI_Status status;
}
else if (id == 2) {
    wtime = MPI_Wtime ();
    output2 = p2_compute_output (input2);
    p2_send_output (output2);
wtime = MPI_Wtime () - wtime;
MPI_Finalize (); }}
```

Teacher's Signature:

Experiment No. 8 Name :

DOMS Page No. 21
Date

AIM

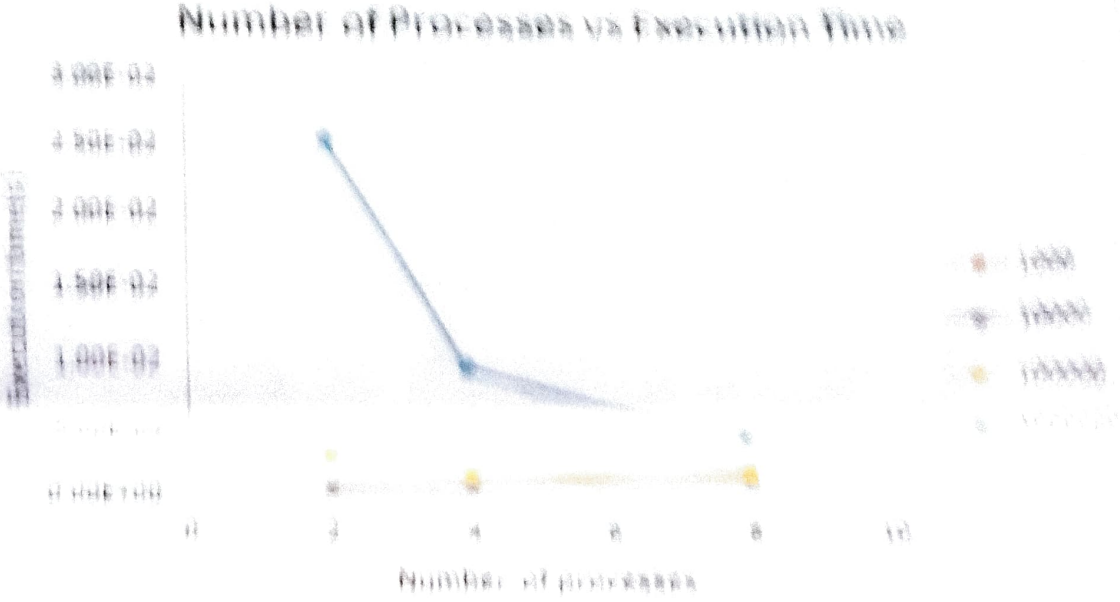
Write MPI-C program which approximates an integral using a quadrature rule.

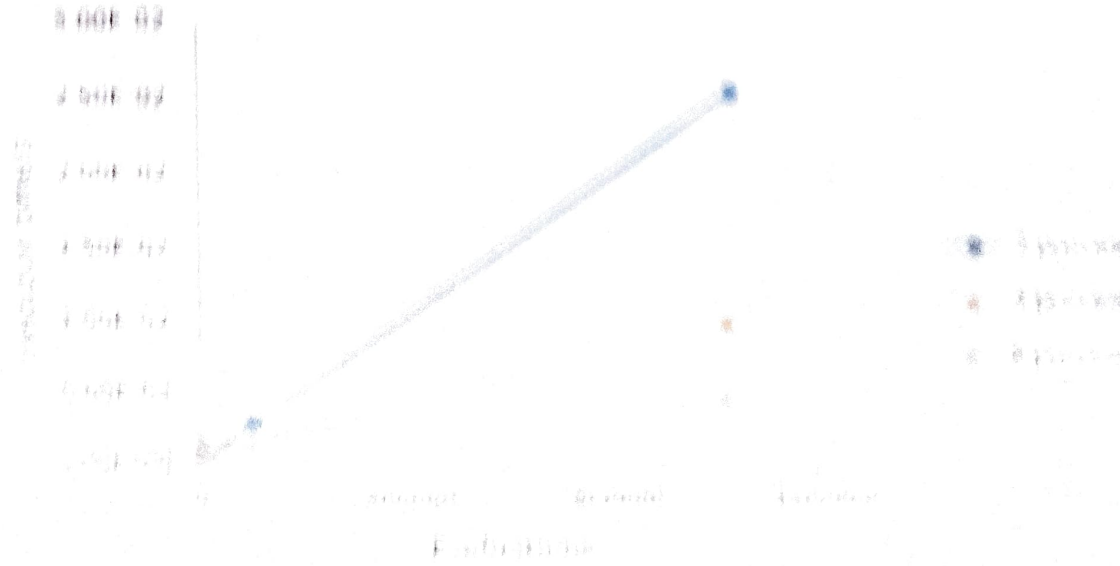
Program

```
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
double f (double x) {
    double pi, value)
    pi = 3.14159265358979793;
    value = 50.0 / (pi * (2500 * x * x + 1.0));
    return value; }
void timestamp (void) {
    #define TIME_SIZE 40
    static char time_buffer [TIME_SIZE];
    const struct tm *tm;
    time_t now; now = time (NULL));
    tm = localtime (&now);
    strftime (time_buffer, TIME_SIZE, "%d %B %Y
    %I %M %S %p", tm);
    printf ("%s \n", time_buffer);
    return;
```

Teacher's Signature:

Evaluations	Execution Time		
	2	4	8
1000	5.50E-04	6.68E-04	1.90E-03
10000	8.08E-04	2.50E-03	3.43E-03
100000	0.003111	0.0016008	0.0013599
1000000	0.0255	0.0102281	0.0134599





Experiment No. 8 Name :

DOMS Page No. 22
Date

```
#undef TIME_SIZE }

int main (int argc, char *argv[]) {
  double a,b, error, exact, my_total, total, wtime, x;
  int i, master = 0, my_id, my_n, n, p, p_num, src;
  double my_a, my_b; MPI_Status status;
  int tag, target; a = 0; b = 10; n = 10000000;
  exact = 0.499363381076454567464;
  MPI_Init (&argc, &argv);
MPI_Comm_Rank ( MPI_COMM_WORLD, &my_id);
MPI_Comm_size (MPI_COMM_WORLD, &p_num);
if (my_id == master) {
        my_n = n/ (p_num-1);
      n = (p_num-1) * my_n; wtime = MPI_Wtime();
    printf ("QUAD MPI - C/MPI version \n");
    printf ("A = %f \n", a); printf ("B = %f \n", b);
}
src = master; MPI_Bcast (&my_n, 1, MPI_INT, src, MPI_COMM_WORLD)
if (my_id == master) {
    for (p=1; p<= p_num-1; p++) {
        my_a = ((double) (p_num-p) *a +
          (double) (p-1) *b / (double) (p_num-1);
  target = p; tag = 1;
  MPI_Send (&my_a, 1, MPI_DOUBLE, target, tag,
      MPI_COMM_WORLD);
```

Teacher's Signature:

```
target=p; tag =2;
MPI_Send (&my_b, 1, MPI_DOUBLE, target, tag,
    MPI_COMM_WORLD); }
total = 0.0; my_total = 0.0; }
else { source = master;
    tag =1; MPI_Recv (&my_a, 1, MPI_DOUBLE,
    source, tag, MPI_COMM_WORLD, &status);
    my_total = 0.0;
    for ( i=1; i<= my_n; i++) {
        x= ((double) (my_n-i) * my_a +
            (double) (i-1) * my_b) / double (my_n-1);
        my_total = my_total + f(x); }
my_total = (my_b-my_a) * my_total / (double) my_n
    MPI_Reduce (&my_total, &total, 1, MPI_Double,
        MPI_SUM, master, MPI_COMM_WORLD);
if (my_id == master) { error = fabs (total - exact);
    utime = MPI_Wtime () - utime; }

MPI_Finalize ();
if (my_id == master)
    printf ("End of Execution");
return 0;
}
```

Teacher's Signature: _____

Experiment No. 9 Name :

DOMS Page No. 24
Date / /

AIM

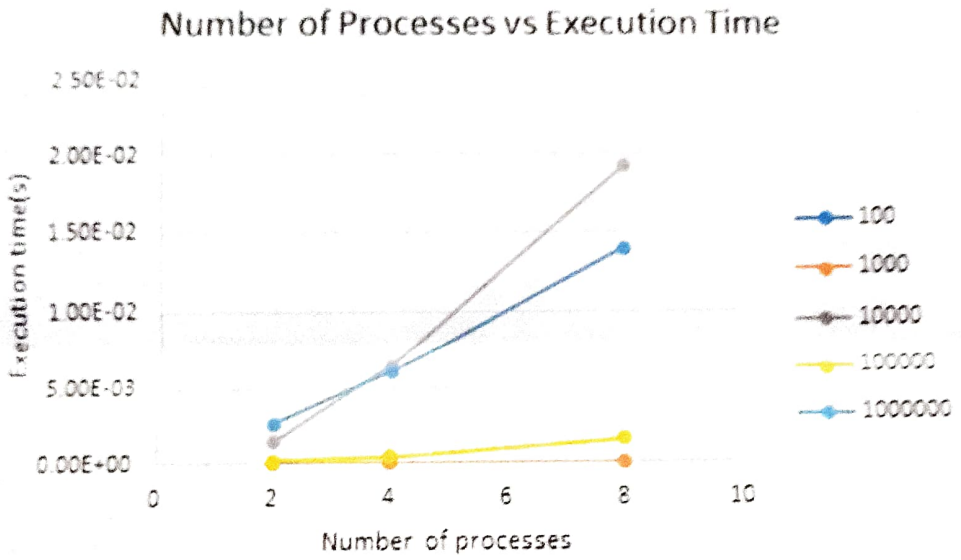
Write a MPI-C program which estimates the time it takes to send a vector of N double precision values through each process in a ring.

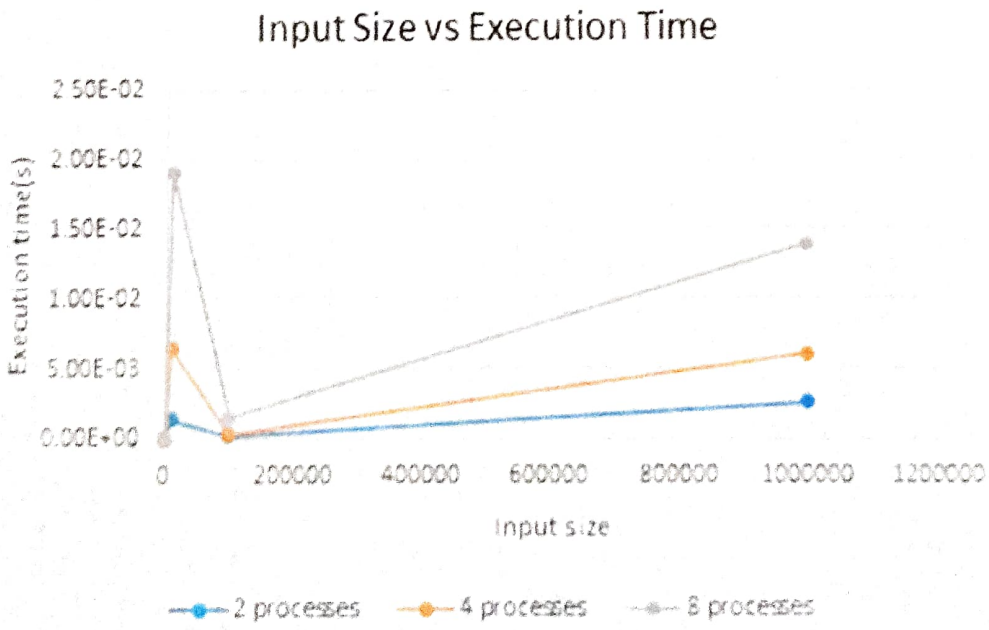
Program

```
#include <mpi.h>
#include <stdlib.h>
#include <stdio.h>
void ring_io (int p, int id)
int main (int argc, char * argv[]) {
  int error, p, id;
  MPI_Init (&argc, &argv)
  MPI_Comm_size (MPI_COMM_WORLD, &p);
  MPI_Comm_rank (MPI_COMM_WORLD, &id);
  if (id == 0) {
    printf ("\n Ring MPI : \n");
    printf (" A ring of processes \n");
    printf ("\n");
    printf ("The no. of processes is %d \n", p);
}
ring_io (p, id);
```

Teacher's Signature:

	Execution Time		
Input Size	2	4	8
100	5.27E-05	1.67E-05	6.29E-05
1000	1.07E-05	1.48E-05	6.76E-05
10000	0.001448	0.006482	0.019143
100000	0.000243	0.000422	0.001558
1000000	0.002617	0.006074	0.013918





Experiment No. 9 Name

DOMS Page No. 25
Date

```
MPI_Finalize();
if (id == 0) {
    printf("Ring MPI:\n");
}
return 0;
}
void ring_io(int p, int id) {
int dest, i, j, n;
int n_test[5] = {100, 1000, 10000, 100000};
int n_test_num = 5, source;
MPI_Status status; double tave;
int test, test_num = 10; double tmax, tmin, wtime;
double *x; if (id == 0) {
printf("Timings based on experiments");
printf("\tN \tT Tave \t Tmax \n"); }

for (i=0; i<n; i++) { n = n_test[i];
x = (double *x) malloc (n * sizeof (double));
if (id == 0) { dest = 1; source = p-1;
    tave = 0.0; tmin = 1.0E+30; tmax = 0.0;
for (test=1; test <= min; test++) {
        for (j=0; j<n; j++)
            { x[j] = (double) (test + j);
    wtime = MPI_Wtime() - wtime;
    tave = tave + wtime;
if (wtime < tmin) tmin = wtime;
```

Teacher's Signature:

```
    if (tmax < wtime) tmax = wtime;
}
tave = tave / (double) (test_num); }
else
{   source = id - 1;
    dest = ((id + 1) % p);
for (test = 1; test <= test_num; test++) {
MPI_Recv (x, n, MPI_DOUBLE, source, 0, MPI_COMM_WORLD,
        &status);
MPI_Send (x, n, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
    }}
free(x); } return; }
```

Teacher's Signature:

Experiment No. 10 Name :

DOMS Page No. 27
Date

AIM

Write an OpenACC program that computes a simple matrix-matrix multiplication using dynamic memory allocation.

Program

```
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>
#define MAX 1000
int size;
double **a, **b, **c, **d;
int main (int argc, char *argv[]){
    SIZE = atoi(argv[1]);
a = (double **) malloc (SIZE * sizeof (double));
b = (double **) malloc (SIZE * sizeof (double));
c = (double **) malloc (SIZE * sizeof (double));
d = (double **) malloc (SIZE * sizeof (double));
int i, j, k; struct timeval, tim;
double t1, t2, tmp;
for (i=0; i<SIZE; ++i){
    a[i] = (double *) malloc (SIZE * sizeof (double));
    b[i] = (double *) malloc (SIZE * sizeof (double));
    d[i] = (double *) malloc (SIZE * sizeof (double));
```

Teacher's Signature:

```
for ( j=0; j< SIZE; ++j) {
  a[i][j] = (double) (i+j);
  b[i][j] = (double) (i-j);
  c[i][j] = 0.0f;
  d[i][j] = 0.0f; }}
for (i=0; i< SIZE; ++i) {
  for (j=0; j<SIZE; ++j) {
    tmp = 0.0f;
    for (k=0; k<SIZE; ++k)
      tmp += a[i][k] * b[k][j];
  d[i][j] = tmp; }}

gettimeofday (&tim, NULL);
t1 = tim.tv_sec + (tim.tv_usec/1000000.0);
#pragma acc data copyin (a,b) copy(c)
#pragma acc kernels
#pragma acc loop tile (1000,1000)
  for (i=0; i<SIZE; ++i) {
    for (j=0; j<SIZE; ++j) {
      tmp = 0.0f;
      for (k=0; k<SIZE; ++k)
      tmp += a[i][k] * b[k][j];
      c[i][j] = tmp; }}
```

Teachers Signature:

```
gettimeofday (&tim, NULL);
t2 = tim.tv_sec + (tv_usec/1000000.0);
for (i=0; i<SIZE; ++i)
    for (j=0; j<SIZE; ++j)
        if (c[i][j] != d[i][j]) {
            exit (1); }

return 0;
}
```

Teacher's Signature:

Experiment No. 11 Name :

DOMS Page No. 30
Date

AIM

Write an OpenACC program to implement two dimensional Jacobi Heration.

Program

```
#include <math.h>
#include <string.h>
#include <openacc.h>
#include <sys/time.h>
#include <stdio.h>
#define NN 1024
#define NM 1024
float A[NN][NM], Anew[NN][NM];

int main(int argc, char** argv) {
    int i, j; const int n = NN; const int m = NM;
    iter_max = 1000; const double tol = 1.0e-6;
    double error = 1.0, t1, t2;
    struct timeval, tim;
    memset(A, 0, n * m * sizeof(float));
    memset(Anew, 0, n * m * sizeof(float));
    for (j=0; j<n; j++) { A[j][0] = 1.0;
        Anew[j][0] = 1.0; }
```

Teacher's Signature:

Experiment No. 11 Name :

```
gettimeofday (&tim, NULL);
t1 = tim.tv_sec + (tm.tv_usec / 1000000.0);
int iter = 0;
#pragma acc data copy(A) create (Anew)
while (error > tol && iter < iter_max){
    error = 0.0;
#pragma acc parallel loop reduction (max : error)
    for (j=1; j<n-1; j++){
        for (i=1; i<m-1; i++){
Anew[j][i] = 0.25 * A[j][i+1] + A[j][i-1]
    + A[j-1][i] + A[j+1][i]);
error = fmax (error, fabs(Anew[j][i] -
A[j][i]));  }}

#pragma acc kernels
    for (j=1; j<n-1; j++){
        for (i=1; i<m-1; i++)
            A[j][i] = Anew[j][i];
    }
if (iter % 100 == 0) printf ("%5d, %0.6f\n", iter);
    iter++; }
gettimeofday ( &tm, NULL);
t2 = tim.tv_sec + (tm.tv_usec / 1000000.0);
return 0; }
```

Teacher's Signature: