Certificate

Name: Aditya Raj Class: 6 'A'

Roll No: 1RV17CS010 Exam No:

Institution RVCE

This is certified to be the bonafide work of the student in the PADP *Laboratory during the academic year* 2020 / 2021 .

No of practicals certified _____ *out of* _____ *in the subject of* _____

.....

Teacher In-charge

.....

Principal

.....

Examiner's Signature

Institution Rubber Stamp

Date: .....

(N.B: The candidate is expected to retain his/her journal till he/she passes in the subject.)

Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
	Open MP				
1.	Monte Carlo	1			
2.	Matrix Multiplication	5			
3.	Sieve of Eratosthenes	9			
4.	Negative of an Image	13			
5.	Points Classification	16			
6.	Word Search	18			
	- MPI -				
7.	Multitasking	21			
8.	Integral	26			
9.	Ring - N double precision values	30			
	- Open ACC -				
10.	Matrix Multiplication	33			
11.	Jacobi Solver	36			

Date _____

Expt. No. 1(a) Page No. 1

PROGRAM - 1(a)

1(a) Write an openMP program that computes the value of PI using Monte-Carlo algorithm.

⇒ Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#define SEED 35791246

int main()
{
        long n=0, i, count=0;
        double x,y,z;
        srand(SEED);
      printf("Size \t\t T1 \t\t\t T2 \t\t T4 \t\t\t T8 \n");
      printf("\t value Time \t\t value Time \t\t value Time
                \t\t ValueTime ");

    for( n=10 ; n<=1000000 ; n*=10)
      {
          printf("\n %d \t",n);
        for(int t=1 ; t<=8 ; t*=2)
          {     count=0;
                double start = omp_get_wtime();
                omp_set_num_threads(t);
        #pragma omp parallel for private(x,y,z) shared(count)
                for(i=0; i<n ; i++)
                  {
```

Teacher's Signature : _____

Date ----------

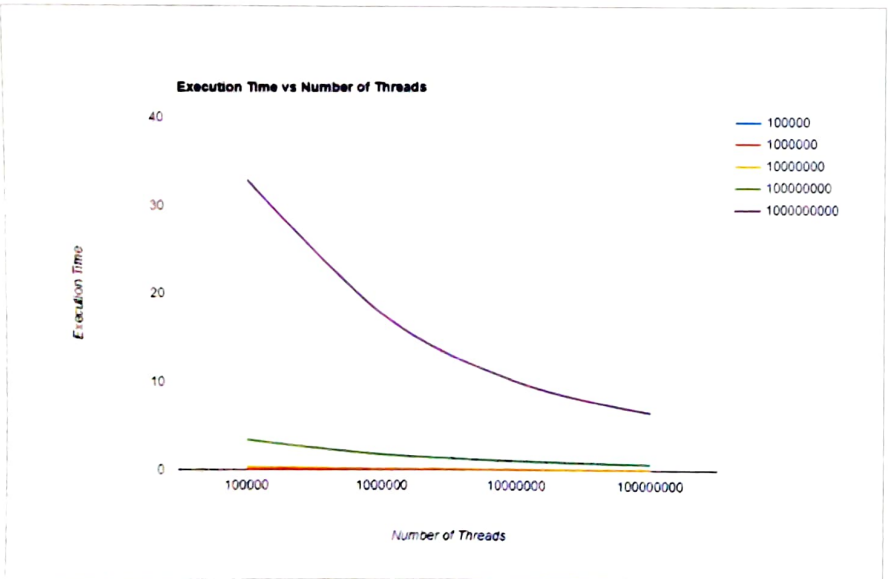
Expt. No. ----------

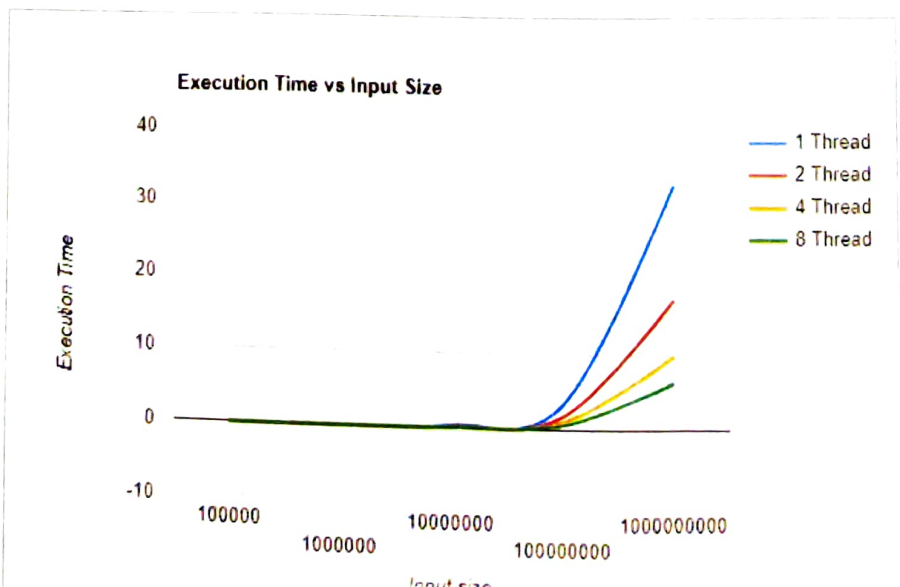
Page No. ----------

```
            x = (double)rand()/RAND_MAX;
            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();
        pf("%lf %lf s\t", pi, stop-start);
    }
}

return 0;
}
```

Teacher's Signature : _____

Input size	Execution Time vs Number of threads							
	1		2		4		8	
	Value	Time	Value	Time	Value	Time	Value	Time
100000	3.14076	0.008000	3.13668	0.005000	3.12744	0.004000	3.1164	0.004000
1000000	3.14234	0.036000	3.14166	0.024000	3.14078	0.014000	3.13622	0.014000
10000000	3.1417	0.334000	3.14145	0.168000	3.14068	0.109000	3.14036	0.069000
100000000	3.14158	3.385000	3.1414	1.724000	3.14121	1.023000	3.14108	0.653000
1000000000	3.14154	32.923000	3.14148	17.479000	3.14146	9.933000	3.14137	6.409000





PROGRAM 1(b)

1b) Write an MPI program that computes the value of PI using Monto-Carlo Algorithm.

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

int main( int argc, char** argv)
 {      int itr = 1000;
        int i, reduced_count, reduced_itr, count = 0;
        int rank;
        double pi, x, y, z, t;
        MPI_Init (&argc, &argv);
        MPI_Comm_rank( MPI_COMM_WORLD, &rank );
        if (rank == 0)
                t = MPI_Wtime();
        if (rank > 0)
          {     srand ( SEED + rank * 10 );
                for (i=0; i<itr; i++)
                {
                    x = (double) rand() / RAND_MAX;
                    y = (double) rand() / RAND_MAX;
                    z = x*x + y*y;
                    if ( z<=1 )
                        count++;
                }
          }
 }
```

Teacher's Signature : _____

Expt. No. ---------- Date ----------

Page No. ----------

```
MPI_Reduce(&count, &reduced_count, 1, MPI_INT, MPI_Sum
            0, MPI_comm_WORLD);
MPI_Reduce( &iter, &reduced_iter, MPI_INT, MPI_SUM,
            0, MPI_comm_WORLD);
reduced_iter = reduced_iter - iter;
if (rank==0)
    {   t = MPI_wtime() - t;
        pi = (double) reduced_count / (double) reduced_iter * 4;
        printf("pi: %lf time: %lf \n", pi, t);
    }
MPI_finalize();
return 0;
}
```

Teacher's Signature : _____

Output

- mpirun -np 2 program1b

  pi: 3.15320 time: 0.000495 s

- mpirun -np 4 program1b

  pi: 3.14053 time: 0.000142 s

- mpirun -np 8 program1b

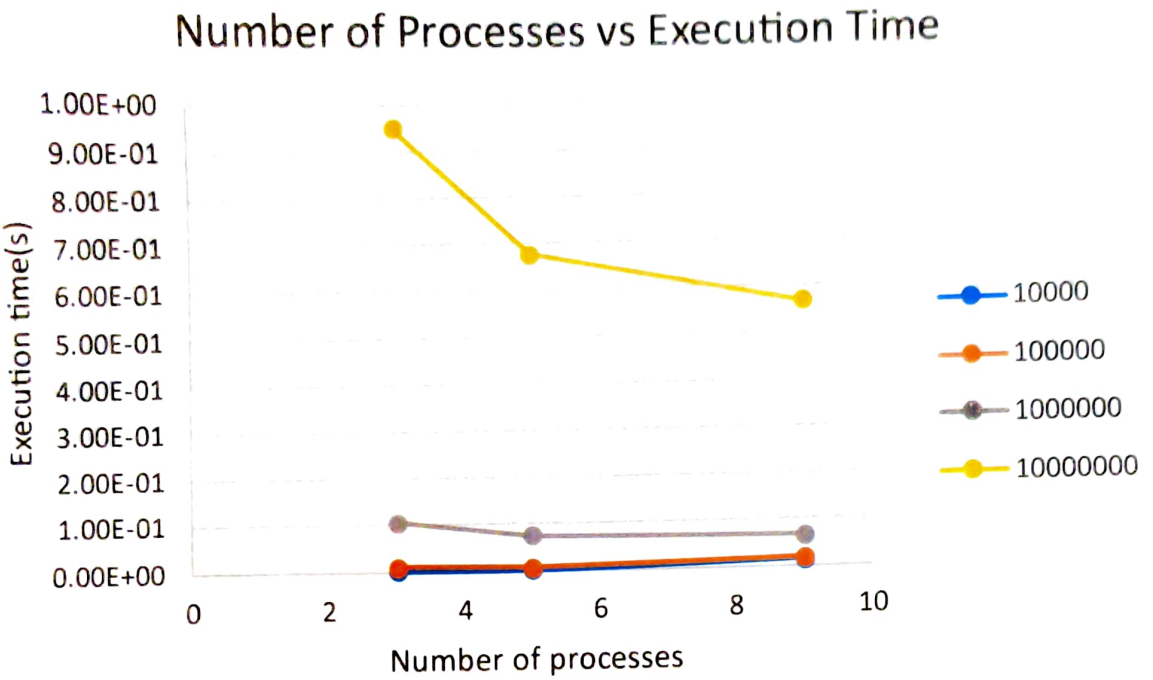
  pi: 3.143714 time: 0.00145 s

- mpirun -np 16 program1b

  pi: 3.1317067 time: 0.003930s

1b

Iterations	Execution Time		
	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


Number of Processes vs Execution Time
1.00E+00
9.00E-01
8.00E-01
7.00E-01
6.00E-01
5.00E-01
4.00E-01
3.00E-01
2.00E-01
1.00E-01
0.00E+00
Execution time(s)
0
2
4
6
8
10
Number of processes
10000
100000
1000000
10000000


Iterations vs Execution Time
1.00E+00
9.00E-01
8.00E-01
7.00E-01
6.00E-01
5.00E-01
4.00E-01
3.00E-01
2.00E-01
1.00E-01
0.00E+00
Execution time(s)
0
5000000
10000000
15000000
Iterations
3 processes
5 processes
9 processes

PROGRAM 2

2) Write an openMP program that computes a simple matrix multiplication using dynamic memory allocation.

a) Illustrate the correctness of the program

```
#include <stdio.h>
int main()
{
    int m,n,p,q,c,d,k,sum = 0;
    int first[10][10], second[10][10], multiply[10][10];

printf("Enter the no. of rows & columns of first matrix\n");
 scanf("%d%d",&m,&n);
printf("Enter elements\n");
 for(c=0; c<m; c++)
        for(d=0; d<n; d++)
            scanf("%d",&first[c][d]);
printf("Enter row and column for second matrix\n");
 scanf("%d%d",&p,&q);
 if(n!=p)
    printf("Matrices cannot be multiplied\n");
else
   { printf("Enter elements of second matrix\n");

    for(c=0; c<p; c++)
        for(d=0; d<q; d++)
            scanf("%d",&second[c][d]);
```

Teacher's Signature : _____

Expt. No. ---------- Date ----------

Page No. ----------

```
        for (c=0; c<m; c++)
        {       for (d=0; d<q; d++)
                {
                        for (k=0; k<p; k++)
                        {
                                sum += first[c][k] * second[k][d];
                        }
                        multiply[c][d] = sum;
                        sum = 0;
                }
        }
printf("Product of entered matrices: \n");
  for(c=0; c<m; c++)
    {   for (d=0; d<q; d++);
                printf("%d\t", multiply[c][d]);
                pri
        printf("\n");
      }
    }
    return 0;
}
```

Teacher's Signature : _____

Output:

- gcc 2a.c -o 2a
- 2a

Enter row & column for first matrix
2 2
Enter the elements of first matrix
2 6
5 4
Enter row & column of second matrix
2 2
Enter the elements of second matrix
3 4
1 5

→ Product of matrices
12 38
19 40

- 2a

Enter no. of rows & column for first matrix
3 3
Enter elements
1 2 3
4 5 6
7 8 9
Enter row & column for second matrix
2 3

⇒ Matrices cannot be multiplied

Date _____

Expt. No. 2(b)

Page No. _____

2(b) Justify the inference when outer 'for' loop is parallelized with & without using the explicit data scope variables.

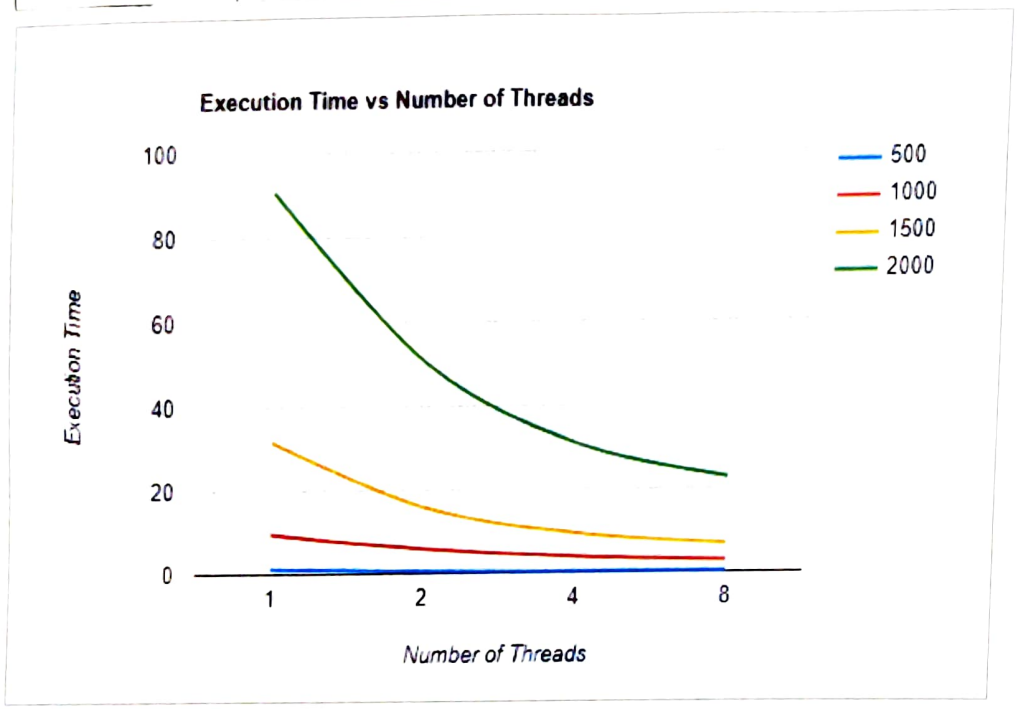
=>

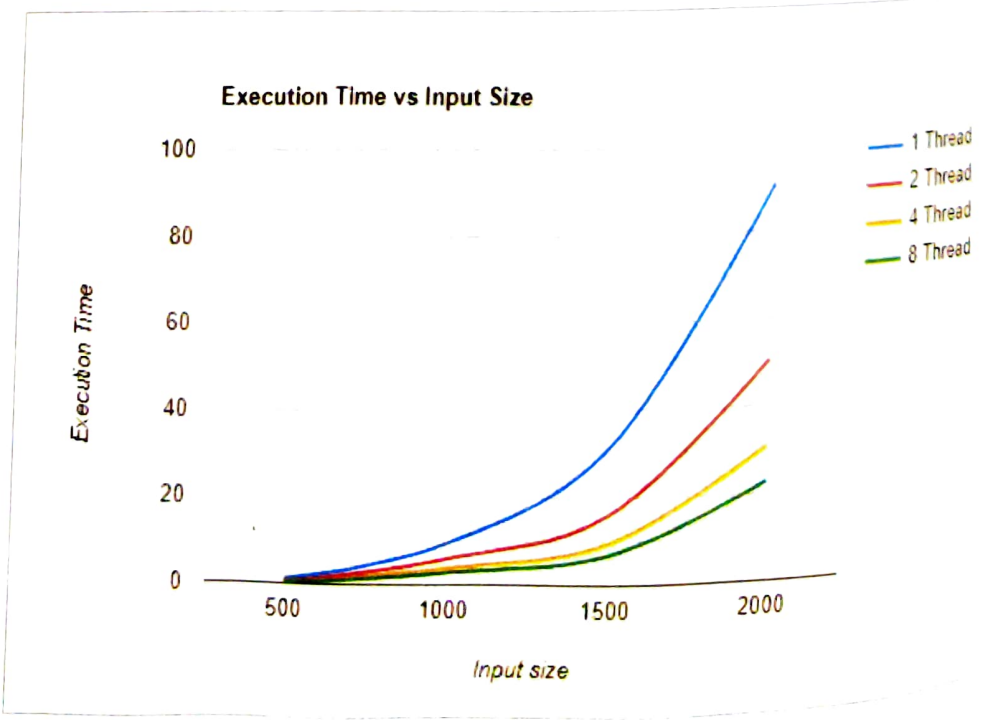
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( int argc, char* argv[] )
{
    int r = atoi( argv[1] ), c = atoi( argv[1] ), i, j, k;
    int count = 0, sum = 0;
    int **arr1 = (int**) malloc ( r * sizeof (int*) );
    for ( int i=0; i<r; i++ )
        arr1[i] = (int*) malloc ( c * sizeof (int) );

    int **arr2 = (int**) malloc ( r * sizeof (int*) );
    for ( i=0; i<r; i++ )
        arr2[i] = (int*) malloc ( c * sizeof (int*) );

    int **arr3 = (int**) malloc ( r * sizeof (int*) );
    for ( i=0; i<r; i++ )
        arr3[i] = (int*) malloc ( c * sizeof (int) );

    for ( i=0; i<r; i++ )
        for ( j=0; j<c; j++ )
            arr1[i][j] = count++;
```

Teacher's Signature : _____

```
    for (i=0; i<r1; i++)
        for (j=0; j<c; j++)
            arr2[i][j] = count++;

double x = omp_get_wtime();
for (i=0; i<r1; i++)
    for (j=0; j<c; j++)
        for (k=0; k<r1; k++)
            arr1[3][i][j] += arr1[i][k] * arr2[k][j];
double y = omp_get_wtime();
printf("Time for %d * %d matrix with 1 thread: ", r1, c);
printf("%lf \n", y-x);

for (int t=2; t<=8; t++)
{   double x = omp_get_wtime();
    omp_set_num_threads(t);
#pragma omp parallel for private(j,k);
    for (i=0; i<r1; j++)
        for (j=0; j<c; j++)
            for (k=0; k<r1; k++)
                arr3[i][j] += arr1[i][k] * arr2[k][j];

double y = omp_get_wtime();
printf("Time for %d * %d matrix with %d threads: ", r1, c, t);
printf("%lf \n", y-x);
}

    return 0;
}
```

Input size	Execution Time-Number of threads			
	1	2	4	8
500	1.247000	0.531000	0.336000	0.296000
1000	9.539000	5.952000	3.864000	2.876000
1500	31.832000	15.949000	9.494000	6.872000
2000	90.786000	50.553000	31.000000	22.856000





PROGRAM 3

3) Write an openmp program for cache friendly sieve of Erotosthenes & cache friendly & parallel Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness.

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

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

long parallelSieve( long n)
  {
        long count = 0;
        long m = (long) sqrt ((double)n);
        long n_factor = 0;
        long* factor = new long[m];

        t = omp_get_wtime();
#pragma omp parallel
        {     bool* Composite = new bool[m+1];
```

Teacher's Signature : _____

Date

Expt. No. Page No.

```
long * strikes = new long[m];

#pragma omp single
{
    memset( composite, 0, m );
    for( long i=2; i<=m; i++ )
        if( !composite[i] )
        {
            ++count;
            strike(composite, 2*i, 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;
        for(long k=0; k<n_factor; k++)
            strikes[k] = (base + factor[k]-1)/
                         factor[k] * factor[k] - base;
    }
    long limit = min(window + m - 1, n) - base;
    for( long k=0; k<n_factor; k++)
        strikes[k] = strike( composite, strikes[k],
                             factor[k], limit ) - m;

    for( long i=0; i<=limit; i++)
        if( !composite[i] ) ++count;
    base += m;
}
```

Teacher's Signature : _____

```
        delete[] striker;
        delete[] composite;
    }
    t += omp_get_wtime() - t;
    delete[] factor;
    return count;
}

long CacheFriendlySieve( long n )
{
    long count = 0;
    long m = (long) sqrt((double)n);
    bool* composite = new bool[n+1];
    memset(composite, 0, n+1);
    long* factor = new long[m];
    long* striker = new long[m];
    long n_factor = 0;

    t = omp_get_wtime();
    for( long i=2; i<=m; i++ )
    {
        if( !composite[i] )
        {
            ++count;
            striker[n_factor] = Strike(composite,
                                       2*i, i, m);
            factor[n_factor++] = i;
        }
    }
}
```

Date

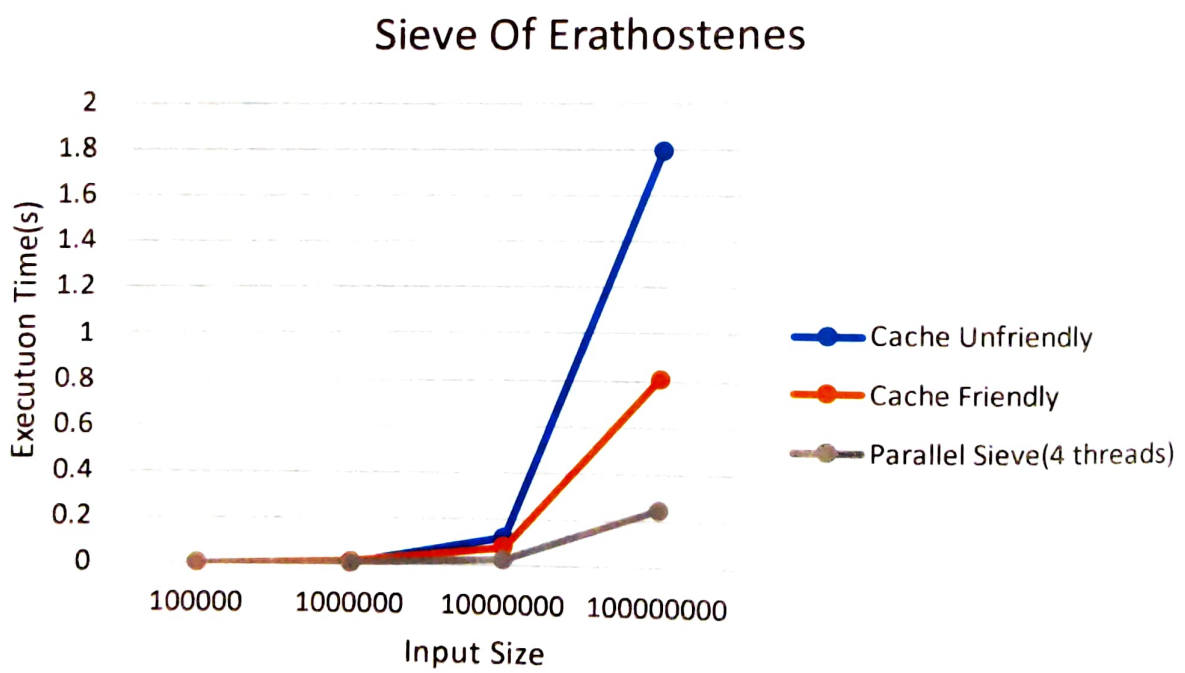
Expt. No. Page No.

```
for( long window = m+1 ; window<=n ; window += m )
    {
        long limit = min ( window + m - 1, n);
      for ( long k=0; k<n_factor ; k++ )
                strikes[k] = Strike ( composite, strikes[k],
                                      factor[k], factor[k], limit);

      for ( long i= window; i<= limit; i++ )
                if ( !composite[i] )
                        ++ count;
    }
      t = omp_get_wtime() - t;
      delete[] strikes;
      delete[] factor;
        delete[] composite;
        return count;
  }
  int main()
{   for ( long i= 100000; i<= 100000000; i= i*10 )
    {
        cout << "\n\n for n = " << i << endl;
        long count = ParallelSieve(i);
        cout << "ParallelSieve \n";
        cout << "Prime count = " << count << "Time: " << t << endl;
        cout << "Cache friendly Sieve \n";
        long count1 = CacheFriendlySieve (i);
        cout << "Prime count = " << count1 << "Time " << t << endl;
    }
  }
```

Teacher's Signature : _____

PROGRAM 3 : Sieve of Eratosthenes

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



PROGRAM 4

4) Write an OpenMP program to convert a colour image to black & white image.

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

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

```
#include<stdio.h>
#include <error.h>
#include< gd.h>
#include <string.h>
#include < omp.h)

int main ( int argc, char * argv[] )
{
    FILE *fp, *fp1 = {0};
    gdImagePtr img;
    char iname[15];
    char oname[15];
    int color, x, y, i = 0;
    int red, green, blue, tmp, tid;
    long w, h;
    Color = x = y = w = h = 0;
    red = green = blue = 0;

Char inputnames[4][15] = { "in1.png", "in2.png", "in3.pnd", "in4.png" };
Char outputnames[4][15] = { "out.png", "out2.png", "out3.png", "out4.png" };
```

Teacher's Signature : _____

Date

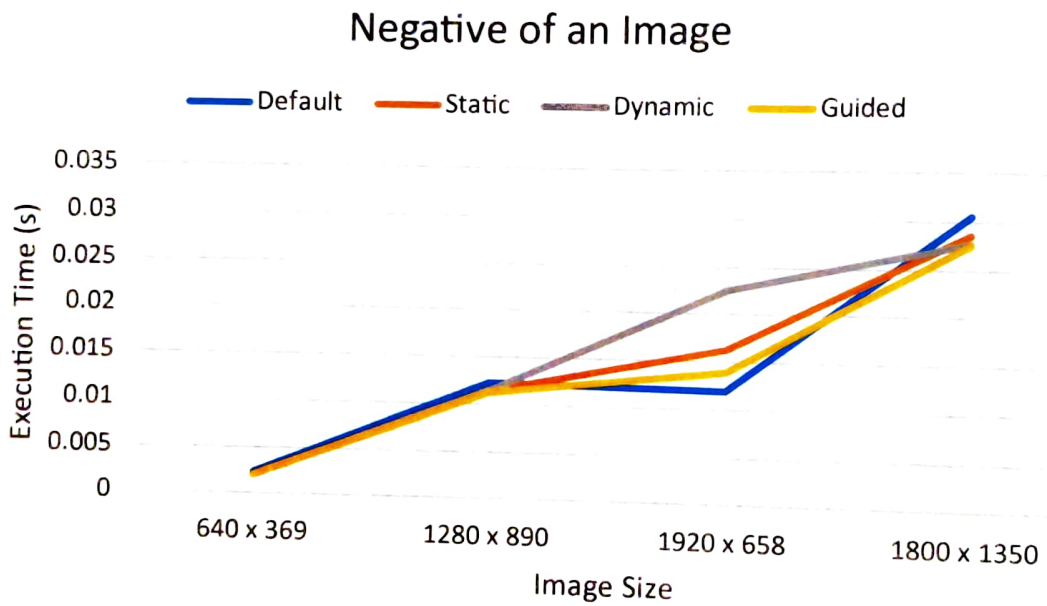
Expt. No. Page No.

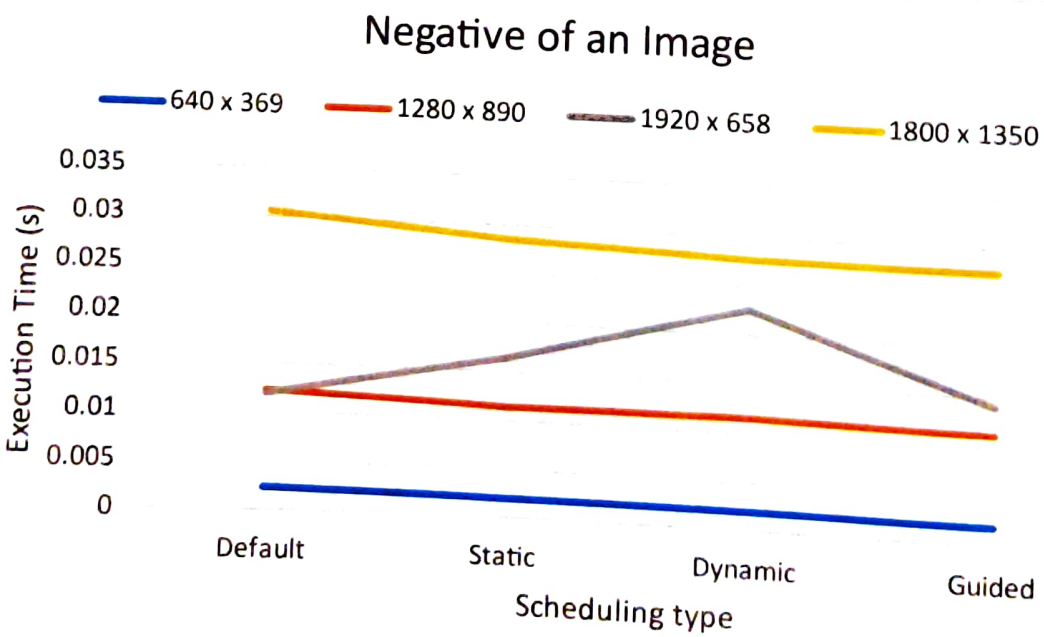
```
omp_sched_t def_sched;
int def_chunk_size;
omp_get_schedule (&def_sched, &def_chunk_size);
pf ("Default %d %d \n", def_sched, def_chunk_size);
pf ("Size \t\t Default \t\t Static \t\t Dynamic \t\t Guided \n");
for (int i=0; i<4; i++)
  {   sprintf (iname, "in%d.png", i+1);
      for (int sched=0x0; 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)
              {   pf ("%ldx%ld \t", w, h);
                  if (i<=1) pf ("\t");
              }
            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++)
                  {   tid = omp_get_thread_num ();
                      color = gdImageGetPixel (img, x, y);
                      red = gdImageRed (img, color);
                      blue = gdImageBlue (img, color);
                      green = gdImageGreen (img, color);
                      tmp = (red + green + blue)/3;
                      red = green = blue = tmp;
```

Teacher's Signature :

```
            color = gdImageColorAllocate(img, red, green, blue);
            gdImageSetPixel(img, x, y, color);
        }
    }
    t = omp_get_wtime() - t;
    fp1 = fopen(oname, "w");
    gdImagePng(img, fp1);
    fclose(fp1);
    gdImageDestroy(imp);
    pf("%.6f\t", t);
}
pf("\n");
return 0;
}
```

Image Size	Scheduling Type			
	Default	Static	Dynamic	Guided
640 x 369	0.002399	0.002054	0.002046	0.002064
1280 x 890	0.012213	0.011349	0.01147	0.011197
1920 x 658	0.011911	0.016365	0.022557	0.013992
1800 x 1350	0.030596	0.028528	0.027761	0.027577





PROGRAM 5

5) Write an openMP parallel program for points classification. Prove the correctness of sequential program with that of parallel.

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

#define CLUSTER_SIZE 4
#define PRINT_POINTS 0

int Cluster[CLUSTER_SIZE][2] = {{75, 25}, {25, 25}, {25, 75},
                                {75, 75}};
long long Cluster_count[CLUSTER_SIZE];
unsigned long points_size[6] = {10000, 500000, 1000000,
                                5000000, 10000000};
short points[10000000][2];
char output[10000] = " ";

void populate_points (unsigned long long size)
{
    long long i;
    for (i=0; i<size; i++)
    {
        srand(1);
        points[i][0] = rand() % 100;
        points[i][1] = rand() % 100;
    }
}
```

Teacher's Signature : _____

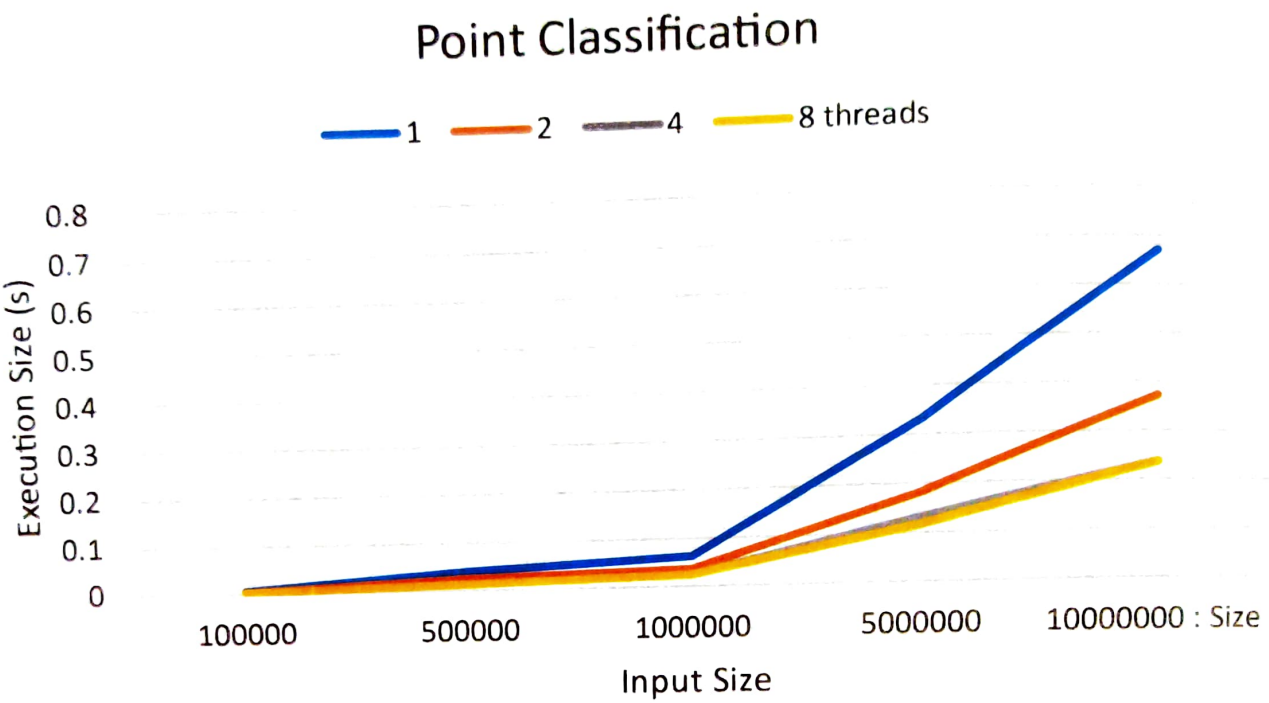
Date

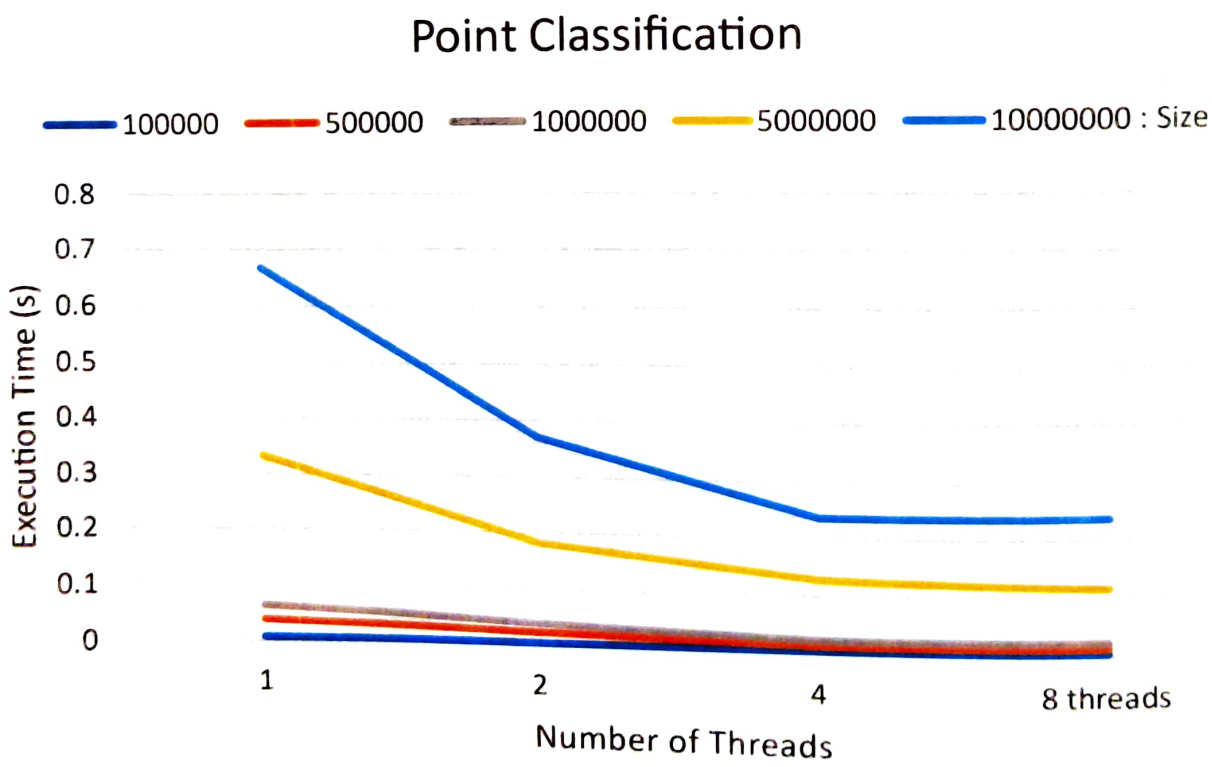
Expt. No. Page No.

```
double get_distance(int x1, int y1, int x2, int y2)
{
    int x = x2-x1; y = y2-y1;
    return (double) sqrt((x*x) + (y*y));
}
int main()
{
    double t;
    sprintf(Output, "Size \t\t 1 thread \t 2 Threads \t 4 Threads \t
                    8 threads \n");
    for(int index=0; index<5; index++)
    {
        pf("Size: %ld, points_size[index]);
        sprintf(Output, "%s%ld \t", Output, points, Sizes[index]);
        if(index!=4)
            sprintf(Output, "%s \t", Output);
        populate_points(point_sizes[index]);
        unsigned long long i;
        If(PRINT_POINTS!=0)
        {
            for(i=0; i< CLUSTER_SIZE; i++)
            { pf("\n Cluster %d : (%d, %d)", i+1, Closer[i][0]
                Cluster[i][i]); }
            pf("\n\n");
        }
int nt=0;
for(nt=1; nt<9; nt*=2)
    {   for(i=0; i< CLUSTER_SIZE; i++)
            Cluster_Count[i]=0;
        t= omp_get_wtime();
```

Teacher's Signature :

Input size	No. of threads			
	1	2	4	8
100000	0.00712	0.003562	0.002419	0.00258
500000	0.039103	0.023072	0.011795	0.011747
1000000	0.063634	0.037818	0.022952	0.023618
5000000	0.332384	0.183873	0.131114	0.11797


Point Classification
1
2
4
8 threads
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Execution Size (s)
100000
500000
1000000
5000000
10000000 : Size
Input Size


Point Classification
100000
500000
1000000
5000000
10000000 : Size
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Execution Time (s)
1
2
4
8 threads
Number of Threads

PROGRAM 6

6) Write an OpenMp program for Word Search in a file & illustrate the performance using different sizes of file.

```
#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 line_c = 0;

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;
    while ( (ch = fgetc(fp)) != EOF && isalpha(ch) == 0)
```

Teacher's Signature : _____

```
    {}
  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_case )
{   int key_index = 0;
    int key_len = strlen(key);
    long word_count = 0;   char ch;
    FILE* fp = fopen ( filename, "r" );
   char temp[40];
    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()
{   int i;
    for( i=0; i< COUNT ; i++ )
                counts[i]=0;
char* my_files = { "file1.txt", "file2.txt", "file3.txt", "file4.txt" };
```

Teacher's Signature : _____

Date

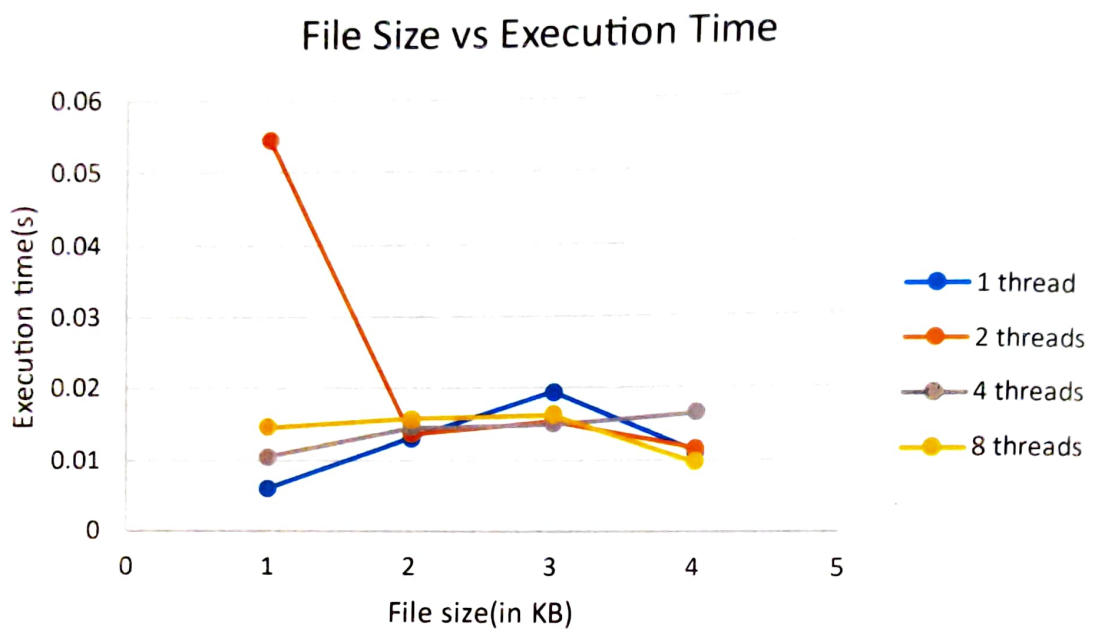
Expt. No.

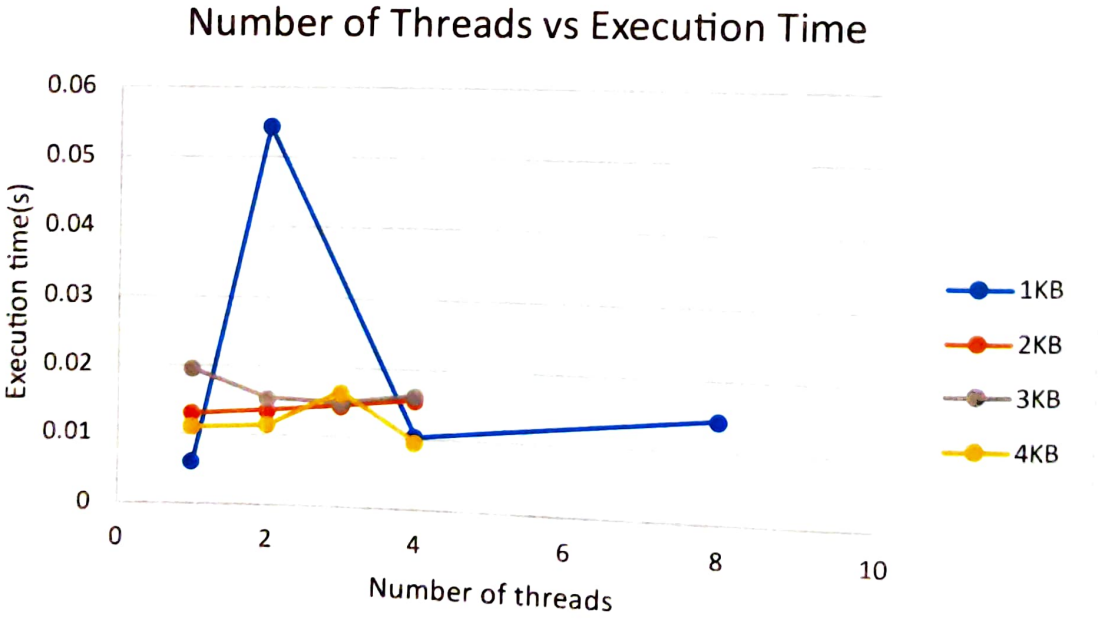
Page No.

```
for (int iter=0; iter<4; i++)
  {   FILE *fp = fopen(my_files[iter], "r");
      fseek(fp, 0L, SEEK_END);
      pf("file Size: %ld KB\n", ftell(fp)/1024);
      fclose(fp);
    for(int t=1; t<=8; t*=2)
      {  omp_set_num_threads(t);
         double start = omp_get_wtime();
         #pragma omp parallel for
             for (i=0; i<COUNT; i++)
                 counts[i]= determine_count(my_files[iter]
                                search_word[i]);
    double time = omp_get_wtime() - start;
     for (i=0; i<COUNT; i++)
          pf("%s: %d ", search_words[i], counts[i]);
    pf("\nTime taken for %d threads: %lf\n", t, time);
  }
}

  return 0;
}
```

Teacher's Signature :

File Size(KB)	Execution Time			
	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


File Size vs Execution Time
Execution time(s)
0.06
0.05
0.04
0.03
0.02
0.01
0
0
1
2
3
4
5
File size(in KB)
1 thread
2 threads
4 threads
8 threads


Number of Threads vs Execution Time
Execution time(s)
0.06
0.05
0.04
0.03
0.02
0.01
0
0
2
4
6
8
10
Number of threads
1KB
2KB
3KB
4KB

PROGRAM 7

7) Write MPI-C Program which demonstrates how to "multitask", to execute several unrelated & distinct tasks simultaneously.

```
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include<stdlib.h>
#include <time.h>

int main (int argc , char *argv[])
  {  int id; int ierr; int input1, input2, output1, output2;
     int p;
     double wtime;
     ierr = MPI_Init( &argc, &argv );
     if (ierr != 0)
       {    pf("MPI_MULTITASK - fatal error! \n");
            pf ("MPI_Init returned non zero ierr \n");
              exit(1);
          }
    ierr = MPI_Comm_rank (MPI_COMM_WORLD , &id);
    ierr = MPI_Comm_Size (MPI_Comm_WORLD, &p);
    if(p<3)
     {    pf("MPI_multitask - fatal error");
          pf("No. of avbl processes must be atleast 3");
           ierr MPI_finalize();
        exit(1);
     }
```

Teacher's Signature : _____

Expt. No. ---------- Date ----------

Page No. ----------

```
if (id==0)
{   timestamp();
    pf("MPI Multitask \n");
      wtime = MPI_wtime();
      p0-set_input (&input1, &input2);
      p0-send_input (input1, input2);
      p0-receive_input( &output1, &output2);
       wtime = MPI_wtime() - wtime;
        pf("Process 0 time = %g", wtime);
     MPI_finalize();
     pf("In MPI_Multitask : In Normal end of execution");
      timestamp();
   }
else if (id==1)
  {     wtime = MPI_wtime();
        input1 = p1_receive_input();
        output1 = p1_compute_output(input1);
        p1_send_output(output1);
        wtime = MPI_wtime() - wtime;
        pf("Process 1 time = %g", wtime);
        MPI_finalize();
   }
else if (id==2)
  {     wtime = MPI_wtime();
        input2 = p2.receive_input();
        output2 = p2.receive_output();
        p2_send_output(output2);
        wtime = MPI_wtime() - wtime;
        pftime("Process 2 time = %g", wtime);
        MPI_finalize();
  }
```

Teacher's Signature : _____

Date .....

Expt. No. .....

Page No. .....

```
Void PO_set_input (int* input1, int * input2)
{
        *input1 = 10000000;
        *input2 = 10000;
    pf("PO_SET_PARAMETERS: \n
      Set_Input = %d Input2 = %d", *input1, *input2);
}

void PO_Send_Input (int input1, int input2)
{
        int id=1, tag=1;
    MPI_Send (&input1, 1, MPI_INIT, tag, MPI_Comm_WORLD);
      id=2, tag=2;
  MPI_Send (&input2, 1, MPI_INIT, tag, MPI_Comm_WORLD);
}

Void PO_receive_output (int * output1, int * output2)
{  int output, output_received=0, source;
     MPI_Status status;
    while(output_received<2)
    { MPI_recv (&Output, 1, MPI_INIT, MPI_ANY_SOURCE,
          MPI_ANY_TAG, MPI_Comm_WORLD, &Status);
       Source = status MPI_SOURCE;
     if (source==1)* Output1 = Output;
   else
        *Output2= output
   Output-received++;
}
pf("Process 1 returned %d Process 2 released %d", *output1,
                                          *Output2);
}
```

Teacher's Signature : _____

Date

Expt. No.

Page No.

```
int p1_receive_input()
{
    int id=0, input1, tag=1;
    MPI_Status status;
    MPI_Recv(&input1, 1, MPI_INT, id, tag, MPI_COMM_WORLD,
                                        &status);
    return 1;
}
int p1_Compute&output(int input1)
{
    int i,j,k, output=0;
    for(i=2; i<=input1; i++)
    {
        j=1; k=0;
        while(i<j)
        {
            if((j%2)==0)
                j=j/2;
            else
                j=3*j+1;
            k++;
        }
        if(output1<k) output1=k;
    }
    return output1;
}
int p1_send_output(int output1)
{
    int id=0, tag=3;
    MPI_Send(&output1, 1, MPI_INT, id, tag, MPI_COMM_WORLD);
}

int p2_recv_input()
{
    int id=0, input2, tag=2;
    MPI_Status status;
```

Teacher's Signature :

Date

Expt. No.

Page No.

```
    MPI_Recv( &input2, 1, MPI_INT, id, tag, MPI_Comm_WORLD, &status);
}
int p2_compute_output ( int input2 )
{
    int i, j; output2=0, prime;
    for (i=2; i<= input2; i++)
    {
        prime = 1;
        for (j=2; j<i; j++)
        {
            if ( i % j == 0)
            {
                prime = 0;
                break;
            }
        }
    }
    return output2;
}
void p2_Send_Output (int Output2)
{
    int id=0, tag=4;
    MPI_Send (&Output2, 1, MPI_INT, id, tag, MPI_comm_WORLD);
}
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);
    strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I : %M %S %p",
                                                    tm );
    pf ( "%s \n", time_buffer);
#undef TIME_SIZE);
```

Teacher's Signature : _____

OUTPUT

- gcc prog7.c -o prog7
  mpirun -np 3 prog7

MPI_ Multitask
PO_set Parameters
Set Input1 = 20000000
Set Input2 = 20000

process 2 time = 8.41861

process 1 returned OUTPUT1 = 618
process 2 returned OUTPUT2 = 17984

Process 1 time = 21.6277
Process 0 time = 21.6266

MPI. Multitask
Normal End of Execution

PROGRAM 8

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

```
#include<math.h>
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main( int argc, char* argv[] )
  {     double a,b, error, exact;
        int i; int master=0;
        double my_a, my_b;
       int my_id, my_n;
        double my_total;
      int n,p, p_num, source;
        MPI_Status status;
        int tag target;
        double total, ctime, x;
      a = 0.0;   b = 10.0;  n = 1000000;
      exact = 0.4993633;
      MPI_Init ( &argc, &argv );
       MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
        MPI_Comm_SIZE( MPI_COMM_WORLD, &pnum );
if (my_id == master)
      {     my_n =   n/(p_num-1);
            n =      (p_num-1) * my_n;
```

Teacher's Signature : _____

Date .....

Expt. No. .....

Page No. .....

```
wtime = MPI_Wtime();
    pf("Quad_mpi - C");
pf("Estimate an integral of f(x) from A to B");
    pf("f(x) = 50/(pi * (2500 * x * x+1)) \n");
    pf("A = %f", a);
    pf("B = %f", b);
    pf("N = %d", n);
    pf("EXACT = %24.16f", exact);
    pf("Using MPI");
}
Source = master
MPI_BCast(&my_n, 1, MPI_INT, Source, MPI_COMM_WORLD);
    if (my_id == master)
    {       for (p=1; p<=pnum-1; p++)
            {       my_a = ((double)(p_num - p) * a
                        + (double)(p-1) * b /
                        (double (pnum-1));

                target = p;
                tag = 1;
            MPI_Send(&my_a, 1, MPI_DOUBLE, target, tag,
                        MPI_COMM_WORLD);
    my_b = ((double)(p_num - p-1) * a + (double)(p) * b /
                        (double)(p_num-1);

    target = p;
    tag = 2;
MPI_Send(&my_b, 1, MPI_DOUBLE, target, tag, MPI_COMM_WORLD);
    }

    total = 0.0;
    my_total = 0.0;
}
```

Teacher's Signature : _____

Date .....

Expt. No. ..... Page No. .....

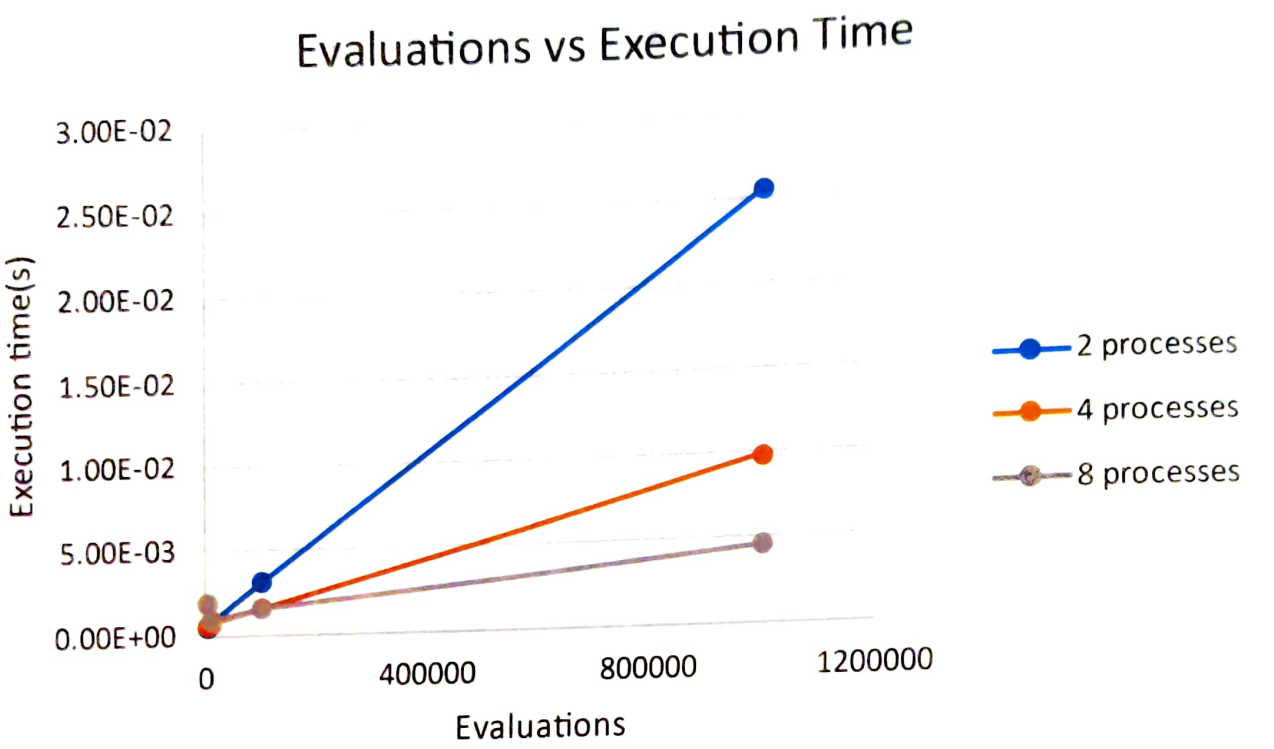
```
else {      source = master;   tag = 1;
    MPI_Recv( &my_a, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD,
              status);
        source = master;
        tag = 2;
    MPI_Recv( &my_b, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD,
              &status);
        my_total = 0.0;
    for ( i = 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);
 pf( "Process %d Contributed MY_TOTAL = %f", my_id, my_total );
  }

MPI_Reduce ( &my_total, &total, 1, MPI_DOUBLE, MPI_SUM, master,
             MPI_COMM_WORLD );
  if (my_id == master)
    {       error = fabs( total - exact );
            wtime = MPI_Wtime() - wtime;
            pf ( "Estimate = %24.16f", wtime );
            pf ( "Error = %e", error );
            pf ( "Time = %f", wtime );
}
```

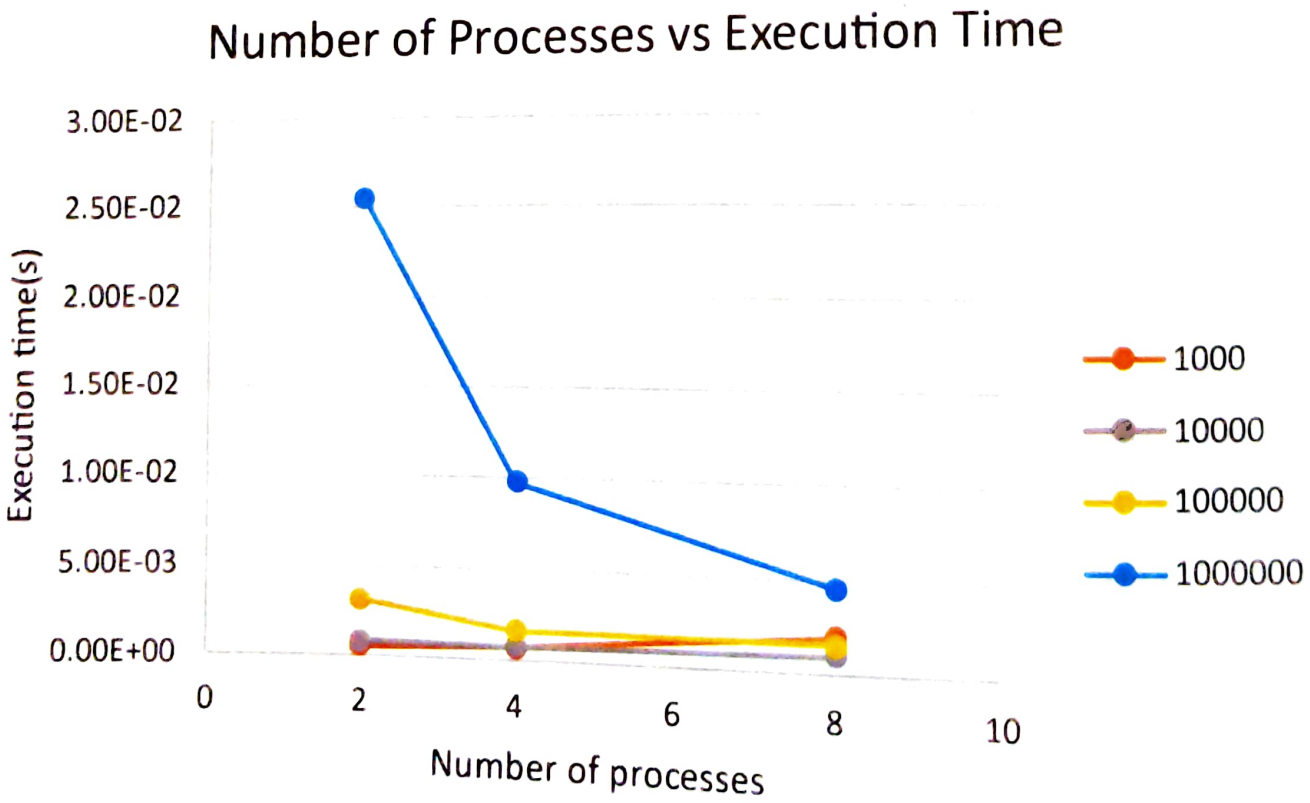
Teacher's Signature : _____

```
MPI_Finalize();
if ( my_id == master )
  { pf( "QUAD-MPI: In Normal end of execution" );
  }
  return 0;
}
double f ( double x )
  {
      double pi, value;
      pi = 3.1415926;
      value = 50.0 / ( pi * ( 2500.0 * x * x + 1.0 ) );
      return value;
  }
```

PROGRAM 8

Evaluations	Execution Time		
	2	4	8
1000	5.50E-04	6.68E-04	1.96E-03
10000	8.08E-04	7.56E-04	9.43E-04
100000	0.003111	0.001608	0.001559
1000000	0.0255	0.009783	0.004566


Evaluations vs Execution Time
3.00E-02
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
Execution time(s)
0
400000
800000
1200000
Evaluations
2 processes
4 processes
8 processes


Number of Processes vs Execution Time
3.00E-02
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
Execution time(s)
0
2
4
6
8
10
Number of processes
1000
10000
100000
1000000

Expt. No. 9 Date

Page No. 30

PROGRAM 9

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

```
#include <mpi.h>
#include <stdlib.h>
#include<stdio.h>

int main( int argc, char* argv[] )
{
    int error, id, p;
    MPI_Init (&argc, &argv);
    MPI_Comm_size( MPI_COMM_WORLD, &p);
    MPI_Comm_Rand( MPI_COMM_WORLD, &id);
    if(id==0)
    {
        pf ("RING-MPI");
        pf ("Measuring time");
        pf ("The no. of process is %d \n", p);
    }
ring_io( p, id);
    MPI_finalize();
    if (id=0)
    {
        pf ("RING MPI");
        pf ("Normal end of execution \n");
    }
return 0;
}
```

Teacher's Signature : _____

Date .....

Expt. No. .....

Page No. .....

```
void ring_io (int p, int id)
{
    int dest, i, j, n;
    int n_test[5] = { 100, 1000, 10000, 100000, 1000000 };
    int n_test_num = 5;
    int source;
    MPI_Status status;
    double tave;
    int test, test_num = 10;
    double tmax, tmin, wtime, *x;
if (id==0)
 { pf("Timings based on %d experiments", test_num);
   pf("N double precision values were sent at process 0");
  pf("Using a total of %d process", p);
   pf("N(size) Tmin Tavg Tmax");
    }
for ( i=0; i<n_test_num; i++)
   {
      n = n_test[i];
    x = (double *) 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<= test_num; test++)
          {   for(j=0; j<n; j++)
              {   x[j] = (double)(test+j); }
    wtime = MPI_wtime();
    MPI_Send ( x, n, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
```

Teacher's Signature : _____

Date .....

Expt. No. .....

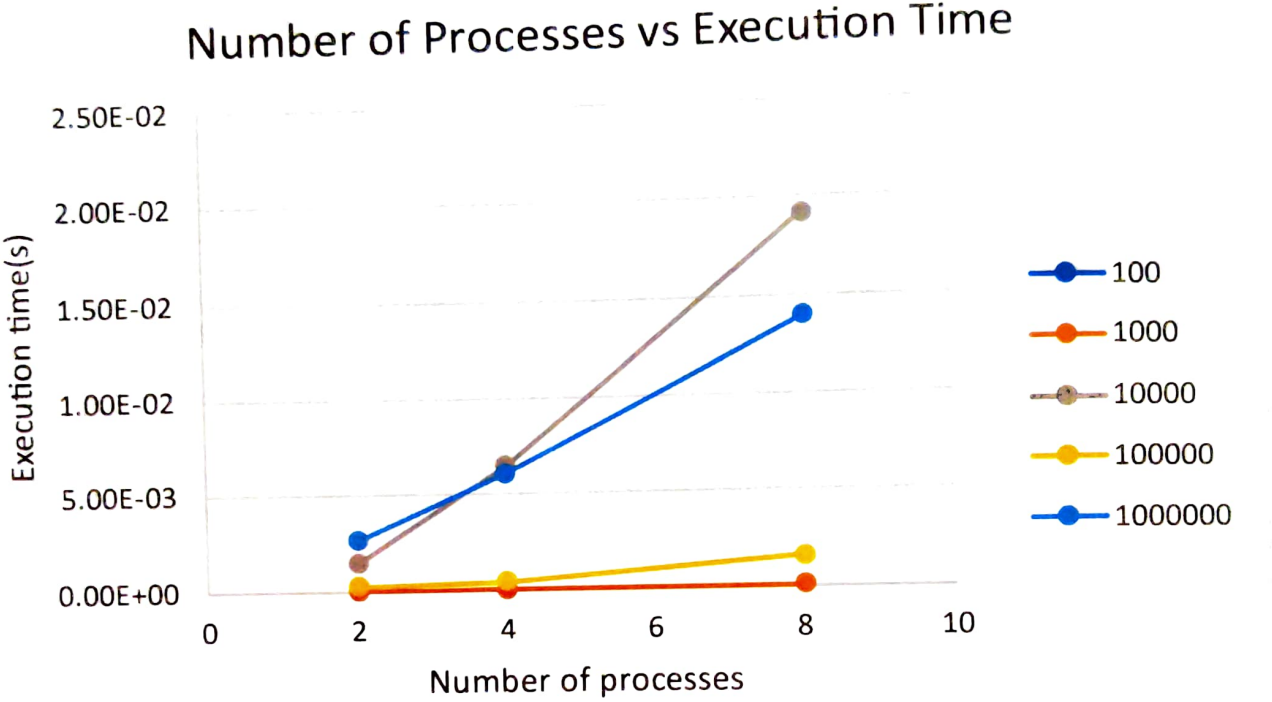
Page No. .....

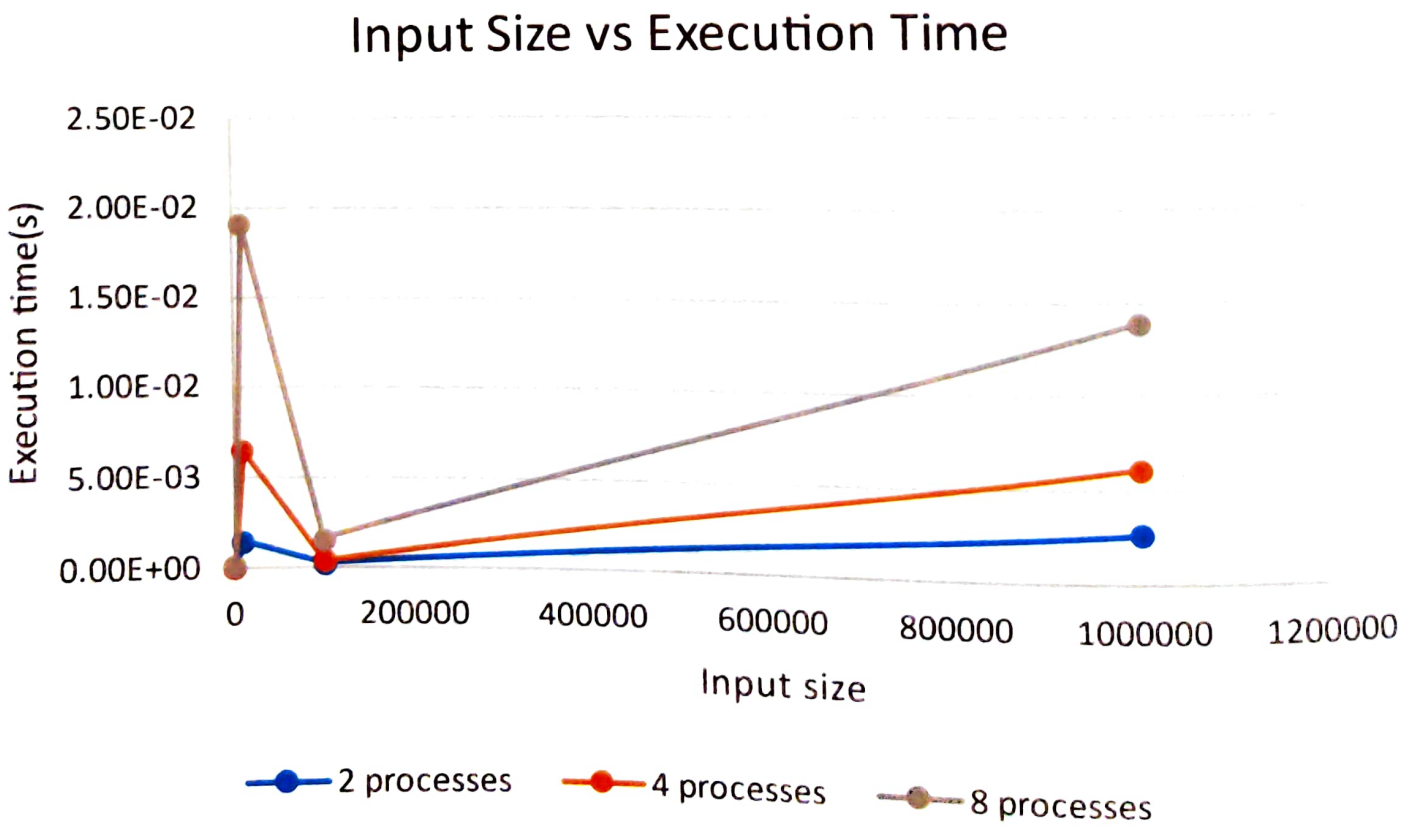
```
MPI_Recv( x, n, MPI_DOUBLE, source, 0, MPI_Comm_WORLD, &Status );
    wtime =     MPI_Wtime( ) - wtime;
        tave =    tave + wtime;
        if ( ~~tmin~~ wtime < tmin )
                {       tmin =  wtime; }
            if ( tmax < wtime )
                    {     tmax =  wtime;
                        }
                }
    tave =    tave / (double) (test_num);
    pf( "%8d %14.6g %14.6g %14.6g", n, tmin, tave, tmax);
}
else
 {      Source  = id - 1;
        dest =    ((id + 1) % p);
       for ( test = 1; test <= test_num; test++ );
          {   MPI_Recv( x, n, MPI_DOUBLE, Source, 0, MA_
                  Comm_WORLD, &status);
       MPI_SEND( x, n, MPI_Double, test, 0, MPI_Comm_WORLD);
        }
 }
    free(x);
}
  return 0;
}
```

Teacher's Signature : _____

9

Input Size	Execution Time		
	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


Number of Processes vs Execution Time
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
Execution time(s)
0
2
4
6
8
10
Number of processes
100
1000
10000
100000
1000000


Input Size vs Execution Time
2.50E-02
2.00E-02
1.50E-02
1.00E-02
5.00E-03
0.00E+00
Execution time(s)
0
200000
400000
600000
800000
1000000
1200000
Input size
2 processes
4 processes
8 processes

Expt. No. 10 Date

Page No. 33

PROGRAM 10

10) Write an OpenAcc program that computes a simple matrix-matrix multiplication using dynamic memory.

```
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>
#define MAX 1000
int Size;
double a[MAX][MAX], b[MAX][MAX], c[MAX][MAX], d[MAX][MAX];

int main()
{
        SIZE = atoi(argv[1]);
        int i,j,k;
        struct timeval tim;
        double t1, t2;
        double tmp;
    for(i=0; i<SIZE; i++)
        {   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++)
```

Teacher's Signature : _____

Date .....

Expt. No. .....

Page No. .....

```
        { 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 (32,32)
        for(i=0; i<SIZE; i++)
            { for(j=0; j<Size; j++)
                { tmp = 0.0f;
#pragma acc loop reduction (+:tmp)
                for(k=0; k<SIZE; k++)
                    { tmp += a[i][k] * b[k][j];
                    }
                c[i][j] = tmp;
                }
        }
gettimeofday(&tim, NULL);
  t2 = tim.tv_sec + (tim.tv_usec/1000000.0);
    pf("for size = %d", SIZE);

pf("%6lf seconds using OpenAcc", t2-t1);

for(i=0; i<SIZE; i++)
    for(j=0; j<SIZE; j++)
        if(c[i][j] != d[i][j])
```

Teacher's Signature : _____

Date _____

Expt. No. _____ Page No. _____

```
{   pf("Error %d %d %f %f", i, j, c[i][j], d[i][j]);
    exit(1);
}
pf("Completed OpenAcc Multiplication");
return 0;
}
```

Teacher's Signature : _____

OUTPUT:

For Size = 100

0.005666 Seconds with openACC
Completed OpenACC matrix multiplication

Size = 200
0.031307 Seconds with OpenACC

Size = 400
0.237965 Seconds with OpenACC

for Size = 800
2.663568 Seconds with OpenACC

PROGRAM 11

Write an OpenAcc program to implement 2-D jacobi

```
#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];
float Anew[NN][NM];

int main(int argc, char** argv)
{
    int i,j;
    const int n = NN;
    const int m = NM;
    const int iter_max = 1000;
    const double tol = 1.0e-6;
    double error = 1.0;
    struct timeval tim;
    double t1,t2;
    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 : _____

Expt. No. ---------- Page No. ----------

```
printf("Jacobi relaxation Calculation %d x %d mesh", n, m);
gettimeofday(&tim, NULL);
int iter = 0;

#pragma acc data copy(A, 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; j++)
    {
        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 parallel loop
for (j = 1; j < n-1; j++)
{   for (i = 1; i < m-1; i++)
    {
        A[j][i] = Anew[j][i];
    }
}
if (iter % 100 == 0)
    pf("%5d, %0.6f", iter, error);
iter++;
}
```

Teacher's Signature : _____

Expt. No. ---------- Date ----------

Page No. ----------

```
        gettimeofDay (&tim, NULL);
    t2 = tim.tv_sec + (tim.tv_usec / 1000000.0);

pf("total : %f s \n", t2-t1);
        return 0;
}
```

Teacher's Signature :

OUTPUT

⇒ gcc -fopenacc jacobi.c -lm
./a.out

Jacobi relaxation Calculation : 1024 × 1024 mesh

0 , 0.25000
100 , 0.002397
200 , 0.001204
300 , 0.000804
400 , 0.000483
500 , 0.000403
600 , 0.000345
700 , 0.000302
800 , 0.000302
900, 0.000269

Total : 15.992072 sec