PARALLEL PROGRAMMING LAB

Course Code : **IS722L** **Credits** : **4:0:0**

Prerequisites: **Microprocessors** **Contact Hours** : **56L**

Course coordinator:JagdeeshSai D

Course Objectives:

- To provide students with contemporary knowledge in parallel and distributed computing
- To equip students with skills to design and analyze parallel and distributed applications.
- To gain hand-on experience with the agent-based and Internet-based parallel and distributed programming techniques.
- Understand, appreciate and apply parallel and distributed algorithms in problem solving.
- Give an overview of approaches to parallelism in functional programming languages in the scientific literature

- **Course Contents:**
- Hello World Program – Basic
- Hello World Program – Modified – Private
- Addition of two array A & B to get array C
- There are two arrays A and B write a program that has two blocks: one for generating array C = A+B and another array D = A*B, such that work in blocks will be done by different threads.
- Example on using critical Directive
- Add two arrays A & B each of 1000 to generate an array C using reduction clause
- Multiply two matrices A & B and find the resultant matrix C
- Write a program to find the number of processes, number of threads etc (Environment information)
- Write a program to find the largest element in an array

- Write a program to find the largest element in an array (usage of locks)
- Write a program to find the sum of an array A
- Write a program to Multiply a matrix by a vector and get the result of the operation
- Write a program to print all the letters of the alphabet A- Z using threads.
- Write a program to show how first private clause works.(Factorial program)
- Write a program to show how last private clause works. (Sum of powers)
- Write a program to find prime numbers (split)
- Write a program for communication among two processes.
- Write a program to demonstrate collective operations in MPI
- Write a program to demonstrate time routines in MPI
- Write a program for Two processes ping pong a number back and forth, incrementing it until it reaches a given value.

Course Outcomes:

- Write, modify and test parallel functional programs, to run on a variety of architectures such as shared memory multiprocessors, networks of commodity servers, and GPUs.
- Identify when using a functional language may be appropriate for solving a parallel programming problem.
- Select an appropriate form of parallel functional programming for a given problem, and explain the choice.
- Ensure numerical accuracy and performance of MPI codes despite communication, synchronization, and replication of computation.
- Write parallel programs that deploy OpenMP and using basic constructs such as parallel blocks and work sharing.

Below are the steps needs tobe followed to run the open mp programs

- Install Ubuntu Version: 12.04 LTS(64 Bit/32 Bit)

- Command line Build instructions for open mp installation
  sudo apt-get install openmpi-bin

- Compile and run the program:
  gcc –fopenmp<name.c>
  ./a.out name

Hint: export OMP_NUM_THREADS=4

1. Hello World Program – Basic

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

int main()
{
#pragma omp parallel
printf("Hello, world! This is thread %d of %d\n", omp_get_thread_num(),
omp_get_num_threads());
}
```

Output:

```
$ gcc -fopenmpmp_hello.c
$ ./a.outmp_hello

Hello, world! This is thread 0 of 4
Hello, world! This is thread 3 of 4
Hello, world! This is thread 2 of 4
Hello, world! This is thread 1 of 4
```

2. Hello World Program – Modified – Private

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

int main(intargc, char *argv[]) {
intiam = 0, np = 1;

#pragma omp parallel default(shared) private(iam, np)
{
#if defined (_OPENMP)
np = omp_get_num_threads();
iam = omp_get_thread_num();
#endif
printf("Hello from thread %d out of %d\n", iam, np);
}
}
```

Output:

```
$ gcc -fopenmp hello2.c
$ ./a.out hello2
Hello from thread 1 out of 4
Hello from thread 2 out of 4
Hello from thread 0 out of 4
Hello from thread 3 out of 4
```

3. Addition of two array A & B to get array C

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNKSIZE 10
#define N 100

int main (intargc, char *argv[])
{
intnthreads, tid, i, chunk;
float a[N], b[N], c[N];

/* Some initializations */

for (i=0; i< N; i++)
a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)
{
tid = omp_get_thread_num();
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}
printf("Thread %d starting...\n",tid);

#pragma omp for schedule (dynamic, chunk)
for (i=0; i<N; i++)
{
c[i] = a[i] + b[i];
printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
}
} /* end of parallel section */
}
```

Output:

```
$ gcc -fopenmpArray_add.c
$ ./a.outArray_add
Thread 2 starting...
Thread 1 starting...
Thread 1: c[10]= 20.000000
Thread 1: c[11]= 22.000000
Thread 1: c[12]= 24.000000
Thread 1: c[13]= 26.000000
Thread 1: c[14]= 28.000000
Thread 1: c[15]= 30.000000
Thread 1: c[16]= 32.000000
Thread 1: c[17]= 34.000000
Thread 1: c[18]= 36.000000
Thread 1: c[19]= 38.000000
Thread 1: c[20]= 40.000000
Thread 1: c[21]= 42.000000
Thread 1: c[22]= 44.000000
Thread 1: c[23]= 46.000000
Thread 1: c[24]= 48.000000
Thread 1: c[25]= 50.000000
Thread 1: c[26]= 52.000000
Thread 1: c[27]= 54.000000
Thread 1: c[28]= 56.000000
Thread 1: c[29]= 58.000000
Thread 1: c[30]= 60.000000
Thread 1: c[31]= 62.000000
Thread 1: c[32]= 64.000000
Thread 1: c[33]= 66.000000
Thread 1: c[34]= 68.000000
Thread 1: c[35]= 70.000000
Thread 1: c[36]= 72.000000
Thread 1: c[37]= 74.000000
Thread 1: c[38]= 76.000000
Thread 1: c[39]= 78.000000
Thread 1: c[40]= 80.000000
Thread 1: c[41]= 82.000000
Thread 1: c[42]= 84.000000
Thread 1: c[43]= 86.000000
Thread 1: c[44]= 88.000000
```

4. There are two arrays A and B write a program that has two blocks: one for generating array C = A+B and another array D = A+B, such that work in blocks will be done by different threads.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 50

int main (intargc, char *argv[])
{

inti, nthreads, tid;
float a[N], b[N], c[N], d[N];

/* Some initializations */

for (i=0; i<N; i++)
{
a[i] = i * 1.5;
b[i] = i + 22.35;
c[i] = d[i] = 0.0;
}

#pragma omp parallel shared (a,b,c,d,nthreads) private(i,tid)
{
tid = omp_get_thread_num();
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Number of threads = %d\n", nthreads);
}
printf("Thread %d starting...\n",tid);

#pragma omp sections nowait
{
#pragma omp section
{
printf("Thread %d doing section 1\n",tid);
for (i=0; i<N; i++)
{
```

```
c[i] = a[i] + b[i];
printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
}
}

#pragma omp section
{
printf("Thread %d doing section 2\n",tid);
for (i=0; i<N; i++)
{
d[i] = a[i] * b[i];
printf("Thread %d: d[%d]= %f\n",tid,i,d[i]);
}
}
} /* end of sectionsnowait */
printf("Thread %d done.\n",tid);
} /* end of parallel section */
}
```

Output:

```
/PP_Lab$ gcc -fopenmp Add_Block_lab4.c
/PP_Lab$ ./a.out Add_Block_lab4
Number of threads = 4
Thread 0 starting...
Thread 0: c[0]= 22.350000
Thread 0: c[1]= 24.850000
Thread 0: c[2]= 27.350000
Thread 0: c[3]= 29.850000
Thread 0: c[4]= 32.349998
Thread 0: c[5]= 34.849998
Thread 0: c[6]= 37.349998
Thread 0: c[7]= 39.849998
Thread 0 doing section 1
Thread 0: c[8]= 42.349998
Thread 0: c[9]= 44.849998
Thread 0: c[10]= 47.349998
Thread 0: c[11]= 49.849998
Thread 0: c[12]= 52.349998
Thread 0: c[13]= 54.849998
Thread 0: c[14]= 57.349998
Thread 0: c[15]= 59.849998

Thread 1 starting...
Thread 1 done.
Thread 3 starting...
Thread 3 done.
Thread 2 starting...
Thread 2 done.
```

5. Example on using critical Directive

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

void A(int* a)
{
printf("a++\n");
   (*a)++;
}

void B(int* a)
{
printf("a--\n");
   (*a)--;
}

void f(int* a)
{
#pragma omp critical
printf("I am Thread %d of %d\n", omp_get_num_thread(),
omp_get_thread_num());

A(a);

sleep(5); /* work done here */

#pragma omp critical
printf("I am Thread %d of  %d\n",omp_get_num_thread(),
omp_get_thread_num());
B(a);
}

int main(intargc, char** argv)
{
inti;
int a = 0;

#pragma omp parallel for
```

```
for(i=0;i<4;i++)
  {
printf("No.of Threads triggerd is %d\n",omp_get_thread_num());
f(&a);
  }
return 0;
}
```

Output:

```
PP_Lab$ gcc -fopenmp critic_lab5.c
PP_Lab$ ./a.out

No.of Threads triggerd is 4
I am Thread 1 of 4
a++
No.of Threads triggerd is 4
I am Thread 2 of 4
a++
No.of Threads triggerd is 4
I am Thread 3 of 4
a++
No.of Threads triggerd is 4
I am Thread 0 of 4
a++
I am Thread 1 of 4
a--
I am Thread 2 of 4
a--
I am Thread 3 of 4
a--
I am Thread 0 of 4
```

6. Add two arrays A & B each of 1000 to generate an array C using reduction clause

/* A reduction clause allows you to get a single value from associative operations such as addition and multiplication. Without the reduction clause, you would need to define a variable as private or shared. If you make the variable private, all copies of it are discarded at the end of the parallel region which means that you can't use their values to complete the sum on exit from the parallel loop. If you declare the variable as shared, all the threads will be reading and writing to the memory address simultaneously which won't work properly for a summation. You will most

likely end up with garbage at the end of the loop. So, by declaring the variable as a reduction, private copies of the variable are sent to each thread and at the end of the parallel region, the copies are summarized (reduced) to give a global shared variable.*/

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

int main (intargc, char *argv[])
{
inti, n;
float a[1000], b[1000], sum;

/* Some initializations */
n = 1000;
for (i=0; i< n; i++)
a[i] = b[i] = i * 1.0;
sum = 0.0;

#pragma omp parallel for reduction(+:sum)
for (i=0; i< n; i++)
sum = sum + (a[i] * b[i]);
printf(" Sum = %f\n",sum);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp reduction_lab6.c
/PP_Lab$ ./a.out reduction_lab6
 Sum = 332833184.000000
```

7. Multiply two matrices A & B and find the resultant matrix C

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define NRA 62 /* number of rows in matrix A */
#define NCA 15 /* number of columns in matrix A */
#define NCB 7 /* number of columns in matrix B */
```

```
int main (intargc, char *argv[])
{
inttid, nthreads, i, j, k, chunk;
double a[NRA][NCA], /* matrix A to be multiplied */
b[NCA][NCB], /* matrix B to be multiplied */
c[NRA][NCB]; /* result matrix C */
chunk = 10; /* set loop iteration chunk size */

/*** Spawn a parallel region explicitly scoping all variables ***/
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
{
tid = omp_get_thread_num();
if (tid == 0)
{
nthreads = omp_get_num_threads();
printf("Starting matrix multiple example with %d threads\n",nthreads);
printf("Initializing matrices...\n");
}

/*** Initialize matrices ***/
#pragma omp for schedule (static, chunk)
for (i=0; i<NRA; i++)
for (j=0; j<NCA; j++)
a[i][j]= i+j;

#pragma omp for schedule (static, chunk)
for (i=0; i<NCA; i++)
for (j=0; j<NCB; j++)
b[i][j]= i*j;


#pragma omp for schedule (static, chunk)
for (i=0; i<NRA; i++)
for (j=0; j<NCB; j++)
c[i][j]= 0;

/*** Do matrix multiply sharing iterations on outer loop ***/
/*** Display who does which iterations for demonstration purposes ***/

printf("Thread %d starting matrix multiply...\n",tid);
#pragma omp for schedule (static, chunk)
```

```
for (i=0; i<NRA; i++)
{
printf("Thread=%d did row=%d\n",tid,i);
for(j=0; j<NCB; j++)
for (k=0; k<NCA; k++)
c[i][j] += a[i][k] * b[k][j];
}
} /*** End of parallel region ***/

/*** Print results ***/
printf("******************************************************\n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++)
{
for (j=0; j<NCB; j++)
printf("%6.2f ", c[i][j]);
printf("\n");
}
printf("******************************************************\n");
printf ("Done.\n");
}
```

Output:

```
/PP_Lab$ gcc -fopenmp matrix_mul_lab7.c
/PP_Lab$ ./a.out matrix_mul_lab7

Starting matrix multiple example with 4 threads
Initializing matrices...
Thread 0 starting matrix multiply...
Thread 2 starting matrix multiply...
Thread=2 did row=20
Thread=2 did row=21
Thread=2 did row=22
Thread=2 did row=23
Thread=2 did row=24
Thread=2 did row=25
Thread=2 did row=26
Thread=2 did row=27
Thread=2 did row=28
Thread=2 did row=29
Thread=2 did row=60
```

```
Thread=2 did row=61
Thread=0 did row=0
Thread=0 did row=1
Thread=0 did row=2
Thread=0 did row=3
Thread=0 did row=4
Thread=0 did row=5
Thread=0 did row=6
Thread=0 did row=7
Thread=0 did row=8

**********************************************************
Result Matrix:
  0.00 1015.00 2030.00 3045.00 4060.00 5075.00 6090.00
  0.00 1120.00 2240.00 3360.00 4480.00 5600.00 6720.00
  0.00 1225.00 2450.00 3675.00 4900.00 6125.00 7350.00
  0.00 1330.00 2660.00 3990.00 5320.00 6650.00 7980.00
  0.00 1435.00 2870.00 4305.00 5740.00 7175.00 8610.00
```

8. Write a program to find the number of processes, number of threads etc (environment information)

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

int main (intargc, char *argv[])
{
intnthreads, tid, procs, maxt, inpar, dynamic, nested;

/* Start parallel region */
#pragma omp parallel private(nthreads, tid)
{
/* Obtain thread number */
tid = omp_get_thread_num();
/* Only master thread does this */
if (tid == 0)
{
printf("Thread %d getting environment info...\n", tid);

/* Get environment information */
procs = omp_get_num_procs();
```

```
nthreads = omp_get_num_threads();
maxt = omp_get_max_threads();
inpar = omp_in_parallel();
dynamic = omp_get_dynamic();
nested = omp_get_nested();

/* Print environment information */
printf("Number of processors = %d\n", procs);
printf("Number of threads = %d\n", nthreads);
printf("Max threads = %d\n", maxt);
printf("In parallel = %d\n", inpar);
printf("Dynamic threads enabled = %d\n", dynamic);
printf("Nested parallelism supported = %d\n", nested);
}
} /* End of parallel */
}
```

Output:

```
/PP_Lab$ gcc -fopenmp Num_pr_thrd_lab8.c
/PP_Lab$ ./a.outNum_pr_thrd _lab8

Thread 0 getting environment info...
Number of processors = 4
Number of threads = 4
Max threads = 4
In parallel = 1
Dynamic threads enabled = 0
Nested parallelism supported = 0
```

9. Write a program to find the largest element in an array

```
#include <stdio.h>
#include <omp.h>
#define MAXIMUM 65536

/* Main Program */
main()
{
int *array, i, Noofelements, cur_max, current_value;
printf("Enter the number of elements\n");
```

```
scanf("%d", &Noofelements);

if (Noofelements<= 0)
{
        printf("The array elements cannot be stored\n");
        exit(1);
}

/* Dynamic Memory Allocation */
array = (int *) malloc(sizeof(int) * Noofelements);
/* Allocating Random Number Values To The Elements Of An Array */
srand(MAXIMUM);
for (i = 0; i<Noofelements; i++)
array[i] = rand();
if (Noofelements == 1)
{
        printf("The Largest Number In The Array is %d", array[0]);
        exit(1);
}
```

/* OpenMP Parallel For Directive And Critical Section **for**: This part of the directive tells the compiler that the next line is a for loop and then it divides the loop into equal parts for each thread, but the order that the loop is performed is nondeterministic, in other words, it does not perform the loop in chronological order.*/

```
cur_max = 0;
omp_set_num_threads(8);

#pragma omp parallel for
for (i = 0; i<Noofelements; i = i + 1)
{
        if (array[i] >cur_max)
        #pragma omp critical
        if (array[i] >cur_max)
        cur_max = array[i];
}

/* Serial Calculation */
```

```
current_value = array[0];
for (i = 1; i<Noofelements; i++)
if (array[i] >current_value)
current_value = array[i];
printf("The Input Array Elements Are \n");
for (i = 0; i<Noofelements; i++)
printf("\t%d", array[i]);
printf("\n");

/* Checking For Output Validity */
if (current_value == cur_max)
        printf("\nThe Max Value Is Same From Serial And Parallel OpenMP
        Directive\n");
else
{
        printf("\nThe Max Value Is Not Same In Serial And Parallel OpenMP
        Directive\n");
        exit(1);
}
/* Freeing Allocated Memory */
printf("\n");
free(array);
printf("\nThe Largest Number In The Given Array Is %d\n", cur_max);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp Largest_lab81.c
/PP_Lab$ ./a.out Largest_lab81
Enter the number of elements
6
The Input Array Elements Are
        553316596    1748907888  680492731    191440832    1061163313
        953167306

The Max Value Is Same From Serial And Parallel OpenMP Directive
The Largest Number In The Given Array Is 1748907888
```

10. Write a program to find the largest element in an array (usage of locks)

```
#include <stdio.h>
#include <omp.h>
#define MINUS_INFINITY -9999
#define MAXIMUM_VALUE 65535

/* Main Program */
main()
{
int *array, i, Noofelements, cur_max, current_value;
omp_lock_t MAXLOCK;
printf("Enter the number of elements\n");
scanf("%d", &Noofelements);

if (Noofelements<= 0)
 {
        printf("The array elements cannot be stored\n");
        exit(1);
}

/* Dynamic Memory Allocation */
array = (int *) malloc(sizeof(int) * Noofelements);

/* Allocating Random Number To Array Elements */
srand(MAXIMUM_VALUE);
for (i = 0; i<Noofelements; i++)
array[i] = rand();
if (Noofelements == 1)
 {
        printf("The Largest Element In The Array Is %d", array[0]);
        exit(1);
}

/* Initializing The Lock */
printf("The locking is going to start\n");
omp_set_num_threads(8);
omp_init_lock(&MAXLOCK);
cur_max = MINUS_INFINITY;
printf("the lock s initialized\n");
/* OpenMP Parallel For Directive And Lock Functions */
```

```
#pragma omp parallel for
for (i = 0; i<Noofelements; i = i + 1)
 {
        if (array[i] >cur_max) {
        omp_set_lock(&MAXLOCK);
        if (array[i] >cur_max)
        cur_max = array[i];
        omp_unset_lock(&MAXLOCK);
}
}

/* Destroying The Lock */
omp_destroy_lock(&MAXLOCK);

/* Serial Calculation */
current_value = array[0];
for (i = 1; i<Noofelements; i++)
if (array[i] >current_value)
current_value = array[i];
printf("The Array Elements Are \n");
for (i = 0; i<Noofelements; i++)
printf("\t%d", array[i]);

/* Checking For Output Validity */
if (current_value == cur_max)
        printf("\nThe Max Value Is Same For Serial And Using Parallel OpenMP
        Directive\n");
else
{
        printf("\nThe Max Value Is Not Same In Serial And Using Parallel
        OpenMP Directive\n");
        exit(1);
}



/* Freeing Allocated Memory */
free(array);
printf("\nThe Largest Number Of The Array Is %d\n", cur_max);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp largest_lab82.c
/PP_Lab$ ./a.out largest_lab82
Enter the number of elements
5
The locking is going to start
the lock s initialized
The Array Elements Are
        842357681    845752218    1085970682  559636718    1183757514
The Max Value Is Same For Serial And Using Parallel openMP Directive

The Largest Number Of The Array Is 1183757514
```

11. Write a program to find the sum of an array A

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

/* Main Program */
doublepartialsum;
/* #pragma ompthreadprivate (partialsum) */

main()
{
double *Array, *Array1, *Check, serial_sum, sum;
intarray_size, i,threadid,tval;

printf("Enter the size of the array\n");
scanf("%d", &array_size);

if (array_size<= 0)
 {
        printf("Array Size Should Be Of Positive Value ");
        exit(1);
}

/* Dynamic Memory Allocation */
Array = (double *) malloc(sizeof(double) * array_size);
Check = (double *) malloc(sizeof(double) * array_size);
```

```
/* Array Elements Initialization */
for (i = 0; i<array_size; i++)
 {
        Array[i] = i * 5;
        Check[i] = Array[i];
}

printf("The Array Elements Are \n");
for (i = 0; i<array_size; i++)
printf("Array[%d]=%lf\n", i, Array[i]);



/* OpenMP Parallel For Directive And Critical Section */
sum=0.0;
omp_set_num_threads(4);
#pragma omp parallel for
for (i = 0; i<array_size; i++)
{
/* printf("the thread num and its iteration is %d %d
\n",omp_get_thread_num(),i); */
#pragma omp critical
sum = sum + Array[i];
}

/* Serial Calculation */
serial_sum = 0.0;
for (i = 0; i<array_size; i++)
serial_sum = serial_sum + Check[i];
printf("the sums are %lf and %lf",sum,serial_sum);

if (serial_sum == sum)
        printf("\nThe Serial And Parallel Sums Are Equal\n");
else
{
        printf("\nThe Serial And Parallel Sums Are UnEqual\n");
        exit(1);
}

/* Freeing Memory */
```

```
free(Check);
free(Array);
printf("\nTheSumOfElements Of The Array Using OpenMP Directives Is
%lf\n", sum);
printf("\nTheSumOfElements Of The Array By Serial Calculation Is %lf\n",
serial_sum);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp sum_lab9.c
/PP_Lab$ ./a.out sum_lab9.c
Enter the size of the array
4
The Array Elements Are
Array[0]=0.000000
Array[1]=5.000000
Array[2]=10.000000
Array[3]=15.000000
the sums are 30.000000 and 30.000000
The Serial And Parallel Sums Are Equal

The SumOfElementsOf The Array Using OpenMP Directives Is 30.000000

The SumOfElementsOf The Array By Serial Calculation Is 30.000000
```

12. Write a program to Multiply a matrix by a vector and get the result of the operation.

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

/* Main Program */
main()
{
intNoofRows, NoofCols, Vectorsize, i, j;
/*float **Matrix, *Vector, *Result, *Checkoutput;*/
double **Matrix, *Vector, *Result, *Checkoutput;
```

```
printf("Read the matrix size noofrows and columns and vectorsize\n");
scanf("%d%d%d", &NoofRows, &NoofCols, &Vectorsize);

if (NoofRows<= 0 || NoofCols<= 0 || Vectorsize<= 0)
{
        printf("The Matrix and Vectorsize should be of positive sign\n");
        exit(1);
}

/* Checking For Matrix Vector Computation Necessary Condition */
if (NoofCols != Vectorsize)
{
        printf("Matrix Vector computation cannot be possible \n");
        exit(1);
}

/* Dynamic Memory Allocation And Initialization Of Matrix Elements */
/* Matrix = (float **) malloc(sizeof(float) * NoofRows); */

Matrix = (double **) malloc(sizeof(double) * NoofRows);
for (i = 0; i<NoofRows; i++)
 {
/* Matrix[i] = (float *) malloc(sizeof(float) * NoofCols); */
Matrix[i] = (double *) malloc(sizeof(double) * NoofCols);
for (j = 0; j <NoofCols; j++)
Matrix[i][j] = i + j;
}

/* Printing The Matrix */
printf("The Matrix is \n");
for (i = 0; i<NoofRows; i++)
{
for (j = 0; j <NoofCols; j++)
printf("%lf \t", Matrix[i][j]);
printf("\n");
}
printf("\n");

/* Dynamic Memory Allocation */
/*Vector = (float *) malloc(sizeof(float) * Vectorsize);*/
Vector = (double *) malloc(sizeof(double) * Vectorsize);
```

```
/* vector Initialization */
for (i = 0; i<Vectorsize; i++)
Vector[i] = i;
printf("\n");

/* Printing The Vector Elements */
printf("The Vector is \n");
for (i = 0; i<Vectorsize; i++)
printf("%lf \t", Vector[i]);

/* Dynamic Memory Allocation */
/* Result = (float *) malloc(sizeof(float) * NoofRows);
Checkoutput = (float *) malloc(sizeof(float) * NoofRows); */
Result = (double *) malloc(sizeof(double) * NoofRows);
Checkoutput = (double *) malloc(sizeof(double) * NoofRows);
for (i = 0; i<NoofRows; i = i + 1)
{
Result[i]=0;
Checkoutput[i]=0;
}



/* OpenMP Parallel Directive */
omp_set_num_threads(32);
#pragma omp parallel for private(j)
for (i = 0; i<NoofRows; i = i + 1)
for (j = 0; j <NoofCols; j = j + 1)
Result[i] = Result[i] + Matrix[i][j] * Vector[j];

/* Serial Computation */
for (i = 0; i<NoofRows; i = i + 1)
for (j = 0; j <NoofCols; j = j + 1)
Checkoutput[i] = Checkoutput[i] + Matrix[i][j] * Vector[j];

/* Checking with the serial calculation */
for (i = 0; i<NoofRows; i = i + 1)
if (Checkoutput[i] == Result[i])
continue;
else
```

```
{
        printf("There is a difference from Serial and Parallel Computation \n");
        exit(1);
}
printf("\nThe Matrix Computation result is \n");
for (i = 0; i<NoofRows; i++)
printf("%lf \n", Result[i]);

/* Freeing The Memory Allocations */
free(Vector);
free(Result);
free(Matrix);
free(Checkoutput);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp mul_matrixvector_lab10.c
/PP_Lab$ ./a.out mul_matrixvector_lab10
Read the matrix size noofrows and columns and vectorsize
3 3
3
The Matrix is
0.000000      1.000000      2.000000

1.000000      2.000000      3.000000

2.000000      3.000000      4.000000


The Vector is
0.000000      1.000000      2.000000

The Matrix Computation result is
5.000000
8.000000
11.000000
```

13. Write a program to print all the letters of the alphabet A- Z using threads.

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

int main(void)
{
inti;
omp_set_num_threads(4);

#pragma omp parallel private(i)
{
// OMP_NUM_THREADS is not a multiple of 26,
// which can be considered a bug in this code.
intLettersPerThread = 26 / omp_get_num_threads();
intThisThreadNum = omp_get_thread_num();
intStartLetter = 'a'+ThisThreadNum*LettersPerThread;
intEndLetter = 'a'+ThisThreadNum*LettersPerThread+LettersPerThread;
for (i=StartLetter; i<EndLetter; i++)
printf("%c", i);
}
printf("\n");
return 0;
}
```

Output:

For 4 Threads:
/PP_Lab$ gcc -fopenmp Alphabet_lab11.c
/PP_Lab$ ./a.out Alphabet_lab11
abcdefstuvwxmnopqrghijkl

For 5 Threads:
/PP_Lab$ gcc -fopenmp Alphabet_lab11.c
/PP_Lab$ ./a.out Alphabet_lab11
fghijpqrstklmnoabcdeuvwxy

For 2 Threads:
/PP_Lab$ gcc -fopenmp Alphabet_lab11.c
/PP_Lab$ ./a.out Alphabet_lab11
Abcdefghijklmnopqrstuvwxyz

14. Write a program to show how thread private clause works.

```
#include <omp.h>

int a, b, i, tid;
float x;

#pragma ompthreadprivate(a, x)
main ()
{
/* Explicitly turn off dynamic threads */
omp_set_dynamic(0);
printf("1st Parallel Region:\n");
#pragma omp parallel private(b,tid)
{
tid = omp_get_thread_num();
a = tid;
b = tid;
x = 1.1 * tid +1.0;
printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x);
} /* end of parallel section */

printf("************************************\n");
printf("Master thread doing serial work here\n");
printf("************************************\n");
printf("2nd Parallel Region:\n");
#pragma omp parallel private(tid)
{
tid = omp_get_thread_num();
printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x);
} /* end of parallel section */
}
```

Output:

```
/PP_Lab$ gcc -fopenmp Private_lab12.c
/PP_Lab$ ./a.out Private_lab12.c
1st Parallel Region:
Thread 3: a,b,x= 3 3 4.300000
Thread 1: a,b,x= 1 1 2.100000
Thread 2: a,b,x= 2 2 3.200000
```

Thread 0: a,b,x= 0 0 1.000000

15. Write a program to show how first private clause works.(Factorial program)

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

longlong factorial(long n)
{
longlongi,out;
out = 1;
for (i=1; i<n+1; i++) out *= i;
return(out);
}

int main(intargc, char **argv)
{
inti,j,threads;
longlong *x;
longlong n=12;

/* Set number of threads equal to argv[1] if present */
if (argc> 1)
{
threads = atoi(argv[1]);
if (omp_get_dynamic())
{
omp_set_dynamic(0);
printf("called omp_set_dynamic(0)\n");
}
omp_set_num_threads(threads);
}

printf("%d threads\n",omp_get_max_threads());
x = (long long *) malloc(n * sizeof(long));
for (i=0;i<n;i++) x[i]=factorial(i);
j=0;
```

```
/* Is the output the same if the following line is commented out? */
#pragma omp parallel for firstprivate(x,j)
for (i=1; i<n; i++)
{
j += i;
x[i] = j*x[i-1];
}
for (i=0; i<n; i++)
printf("factorial(%2d)=%14lld x[%2d]=%14lld\n",i,factorial(i),i,x[i]);
return 0;
}
```

Output:

```
/PP_Lab$ gcc -fopenmp factorial_lab13.c
/PP_Lab$ ./a.out factorial_lab13
1 threads
factorial( 0)=           1 x[ 0]=           1
factorial( 1)=           1 x[ 1]=           1
factorial( 2)=           2 x[ 2]=           3
factorial( 3)=           6 x[ 3]=          18
factorial( 4)=          24 x[ 4]=         180
factorial( 5)=         120 x[ 5]=        2700
factorial( 6)=         720 x[ 6]=       56700
factorial( 7)=        5040 x[ 7]=     1587600
factorial( 8)=       40320 x[ 8]=    57153600
factorial( 9)=      362880 x[ 9]=  2571912000
factorial(10)=     3628800 x[10]= 141455160000
factorial(11)=    39916800 x[11]= 9336040560000
```

16. Write a program to show how last private clause works. (Sum of powers)

```
#include <stdio.h>
#include <malloc.h>
#include <omp.h>
int main(intargc, char **argv)
{
inti,j,threads;
int x[10];
int *sum_of_powers;
int n=10;
```

```
/* Set number of threads equal to argv[1] if present */
if (argc> 1)
{
threads = atoi(argv[1]);
if (omp_get_dynamic())
{
omp_set_dynamic(0);
printf("called omp_set_dynamic(0)\n");
}
omp_set_num_threads(threads);
}
printf("%d threads max\n",omp_get_max_threads());
sum_of_powers = (int *) malloc(n * sizeof(int));

/* Is the output the same if the lastprivate clause is commented out? */
#pragma omp parallel for private(j) lastprivate(x)
for (i=0; i<n; i++)
{
printf("%d threads currently executing\n",omp_get_num_threads());
x[0]  = 1;
for (j=1; j<4; j++) x[j] = x[j-1] * (i+1);
sum_of_powers[i] = x[0] + x[1] + x[2] + x[3];
}
for (i=0; i<n; i++)
{
printf("Sum of powers 0-3 of %d = %d\n",i+1,sum_of_powers[i]);
}
printf("n cubed = %d\n",x[3]);
return 0;
}
```

Output:

```
/PP_Lab$ gcc -fopenmp sum_power_lab14.c
/PP_Lab$ ./a.outsum_power _lab14
1 threads max
1 threads currently executing
1 threads currently executing
1 threads currently executing
1 threads currently executing
1 threads currently executing
1 threads currently executing
1 threads currently executing
```

1 threads currently executing
1 threads currently executing

17. Write a program to find prime numbers (split)

```
#include <stdio.h>
#include <omp.h>
#define N 100000000
#define TRUE 1
#define FALSE 0

int main(intargc, char **argv )
{
char host[80];
int *a;
inti, k, threads, pcount;
double t1, t2;
int found;

/* Set number of threads equal to argv[1] if present */
if (argc> 1)
{
threads = atoi(argv[1]);
if (omp_get_dynamic())
{
omp_set_dynamic(0);
printf("called omp_set_dynamic(0)\n");
}
omp_set_num_threads(threads);
}
printf("%d threads max\n",omp_get_max_threads());
a = (int *) malloc((N+1) * sizeof(int));

// 1. create a list of natural numbers 2, 3, 4, ... none of which is marked.
for (i=2;i<=N;i++) a[i] = 1;

// 2. Set k = 2, the first unmarked number on the list.
k = 2;
t1 = omp_get_wtime();
// 3. Repeat
#pragma omp parallel firstprivate(k) private(i,found)
while (k*k <= N)
```

```
{
// a. Mark all multiples of k between k^2 and N
#pragma omp for
for (i=k*k; i<=N; i+=k) a[i] = 0;
// b. Find the smallest number greater than k that is unmarked
// and set k to this new value until k^2 > N
found = FALSE;
for (i=k+1;!found;i++)
{
if (a[i]){ k = i; found = TRUE; }
}
}
t2 = omp_get_wtime();
printf("%.2f seconds\n",t2-t1);

// 4. The unmarked numbers are primes
pcount = 0;
for (i=2;i<=N;i++)
{
if( a[i] )
{
pcount++;
//printf("%d\n",i);
}
}
printf("%d primes between 0 and %d\n",pcount,N);
}
```

Output:

```
/PP_Lab$ gcc -fopenmp prime_lab15.c
/PP_Lab$ ./a.out prime_lab15
1 threads max
2.70 seconds
5761455 primes between 0 and 100000000
```

COURSE ASSESSMENT – OTHER COMPONENTS

11 a) COURSE ASSESSMENT – RECORD EVALUATION

Record Evaluation is considered for a total of 10 marks. Record valuation is a part of continuous evaluation which will be carried out in each lab on regular basis.

11 b) COURSE ASSESSMENT –VIVA

Viva is considered for a total of 10 marks. Viva is conducted by asking questions to the students orally during lab IA test and is one of the mandatory components during SEE taken by the external examiner.

INDIRECT ASSESSMENT METHOD: COURSE END SURVEY – ANALYSIS

12 a) COURSE END SURVEY QUESTIONS

M S Ramaiah Institute of Technology (Autonomous Institute, Affiliated to VTU) Department of Information Science and Engineering **COURSE END SURVEY**	
USN: Name: Term: **13.08.2015 to 16.12.2015** Credits: 0:0:1 Subject Code: IS702L Subject: Parallel Programming Lab Semester: 7 Section: Date: **13.08.2015 to 16.12.2015**	
CO1	To provide students with contemporary knowledge in parallel and distributed computing
O Excellent O Very Good O Good O Satisfactory O Poor	
CO2	To equip students with skills to design and analyze parallel and distributed applications
O Excellent O Very Good O Good O Satisfactory O Poor	
CO3	To gain hand-on experience with the agent-based and Internet-based parallel and distributed programming techniques
O Excellent O Very Good O Good O Satisfactory O Poor	
CO4	Understand, appreciate and apply parallel and distributed algorithms in problem solving.
O Excellent O Very Good O Good O Satisfactory O Poor	
CO5	Give an overview of approaches to parallelism in functional programming languages in the scientific literature
O Excellent O Very Good O Good O Satisfactory O Poor	

Student Signature:

12 b) RUBRICS EVALUATION OF COURSE END SURVEY

Sl. No.	Parameter	Weightage
1	Excellent	5
2	Very Good	4
3	Good	3
4	Satisfactory	2
5	Poor	1

12 c) COURSE END SURVEY (CES) ANALYSIS

	Parameter	CO1	CO2	CO3	CO4	CO5
SECTION A	Excellent	22	21	19	19	15
	Very Good	11	15	16	18	19
	Good	8	4	6	4	7
	Satisfactory	0	1	0	0	0
	Poor	0	0	0	0	0
Total No. of Students		41	41	41	41	41
Attainment (in percentage)		86.83	87.32	86.34	87.32	83.9

	Parameter	CO1	CO2	CO3	CO4	CO5
SECTION B	Excellent	19	20	17	13	14
	Very Good	14	15	17	16	16
	Good	11	10	12	12	10
	Satisfactory	1	2	1	5	4
	Poor	2	0	0	1	3
Total No. of Students		47	47	47	47	47
Attainment (in percentage)		80	82.55	81.28	74.89	74.47

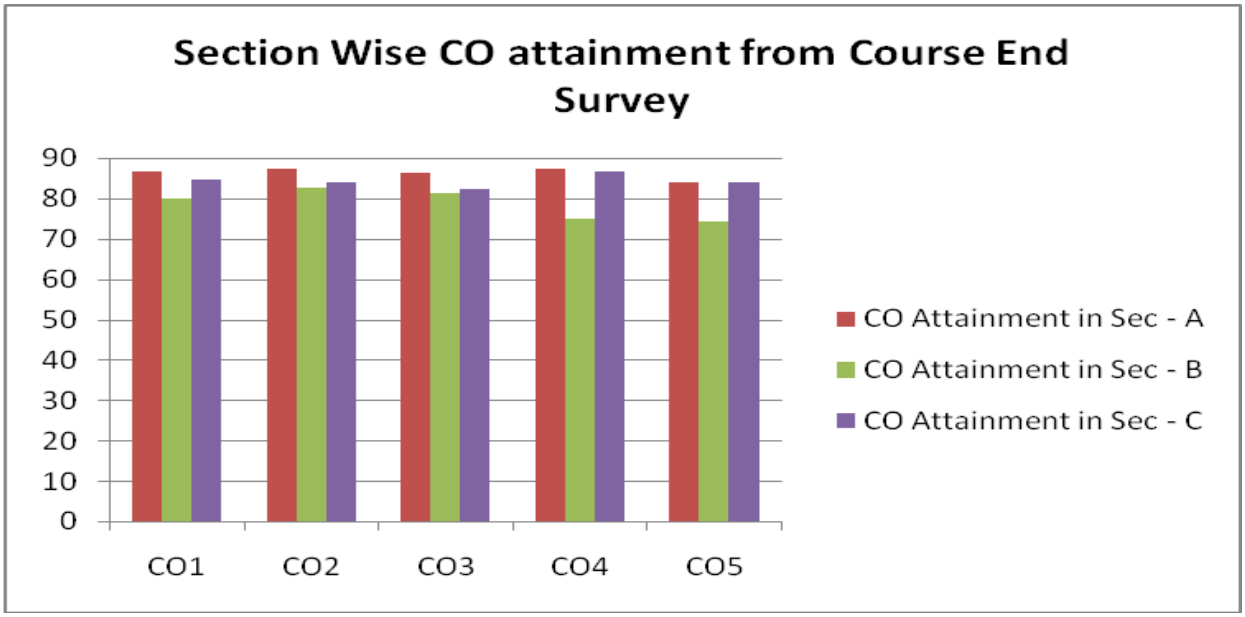
	Parameter	CO1	CO2	CO3	CO4	CO5
SECTION C	Excellent	19	19	20	24	18
	Very Good	22	21	16	17	23
	Good	6	7	10	6	6
	Satisfactory	1	1	2	1	1

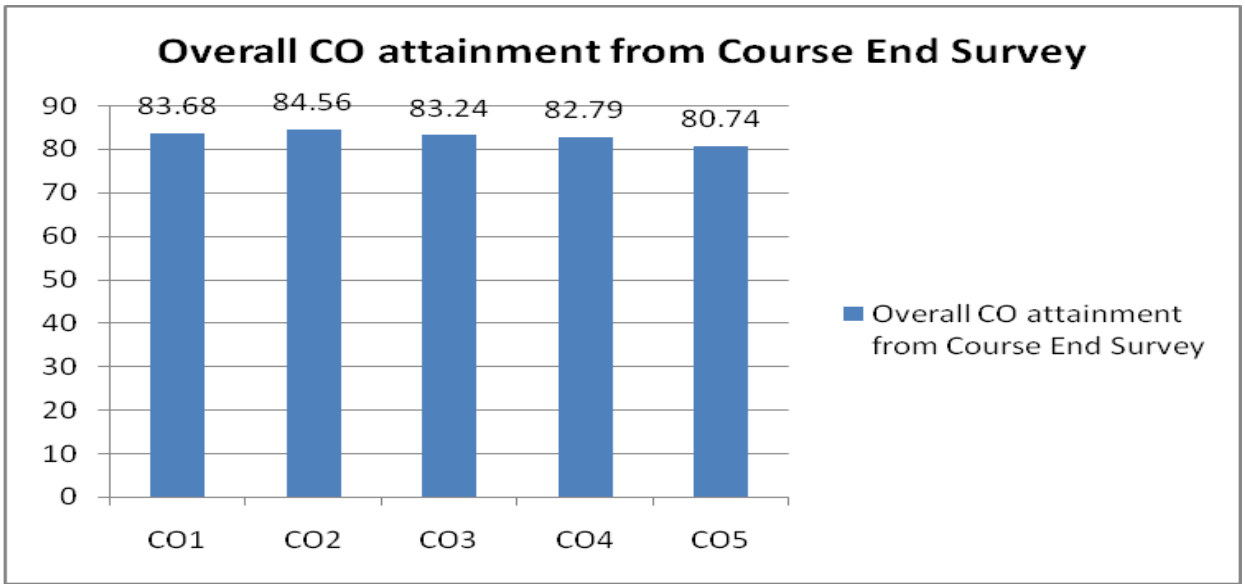
	Poor	0	0	0	0	0
Total No. of Students		48	48	48	48	48
Attainment (in percentage)		84.58	84.17	82.5	86.67	84.17

	Parameter	**CO1**	**CO2**	**CO3**	**CO4**	**CO5**
OVERALL (ALL THREE SECTIONS COMBINED)	Excellent	60	60	56	56	47
	Very Good	47	51	49	51	58
	Good	25	21	28	22	23
	Satisfactory	2	4	3	6	5
	Poor	2	0	0	1	3
Total No. of Students		136	136	136	136	136
Attainment (in percentage)		83.68	84.56	83.24	82.79	80.74

Course Outcome	**CO Attainment from CES for Sec - A**	**CO Attainment from CES for Sec - B**	**CO Attainment from CES for Sec - C**	**Overall CO Attainment from CES**
CO1	86.83	80	84.58	83.68
CO2	87.32	82.55	84.17	84.56
CO3	86.34	81.28	82.5	83.24
CO4	87.32	74.89	86.67	82.79
CO5	83.9	74.47	84.17	80.74

Note: CO Attainment in the above table is measured in percentage (%).




Overall CO attainment from Course End Survey
90
80
70
60
50
40
30
20
10
0
83.68
84.56
83.24
82.79
80.74
CO1
CO2
CO3
CO4
CO5
Overall CO attainment from Course End Survey

COURSE DELIVERY METHODS – ANALYSIS

13 a) COURSE DELIVERY METHODS

The following Course Delivery Methods are planned for this course:

- **Method 1(Chalk & Talk):** This is the traditional teaching method using black-board and chalk.
- **Method 4 (Demonstrations):** This teaching method is best suited for programming courses or courses where software tools can be used. Here practical demonstration of some concepts is done using simulation tools during the practical session.

13 b) RUBRICS EVALUATION OF COURSE DELIVERY METHODS

Sl. No.	Parameter	Weightage
1	Excellent	5
2	Very Good	4
3	Good	3
4	Satisfactory	2
5	Poor	1

13 c) COURSE DELIVERY METHODS (CDM) ANALYSIS

	Parameter	Course Delivery Methods	
		Chalk & Talk	Demonstrations
SECTION A	Excellent	19	20
	Very Good	8	11
	Good	13	6
	Satisfactory	0	3
	Poor	0	0
Total No. of Students		40	40
Attainment (in percentage)		83	84

	Parameter	Course Delivery Methods	
		Chalk & Talk	Demonstrations
SECTION B	Excellent	24	23
	Very Good	8	10
	Good	5	5
	Satisfactory	4	3
	Poor	0	0
Total No. of Students		41	41
Attainment (in percentage)		85.37	85.85

	Parameter	Course Delivery Methods	
		Chalk & Talk	Demonstrations
SECTION C	Excellent	20	18
	Very Good	16	22
	Good	6	6
	Satisfactory	4	0
	Poor	0	0
Total No. of Students		46	46
Attainment (in percentage)		82.61	85.22

	Parameter	Course Delivery Methods	
		Chalk & Talk	Demonstrations
OVERALL (ALL THREE SECTIONS COMBINED)	Excellent	63	61
	Very Good	32	43
	Good	24	17
	Satisfactory	8	6
	Poor	0	0
Total No. of Students		127	127
Attainment (in percentage)		83.62	85.04

The formula to measure CO attainment through In-Direct assessment methods is:

$$\text{CO Attainment} = \frac{((5 * E) + (4 * VG) + (3* G) + (2* S) +(1 * P)) X100}{(N X 5)}$$

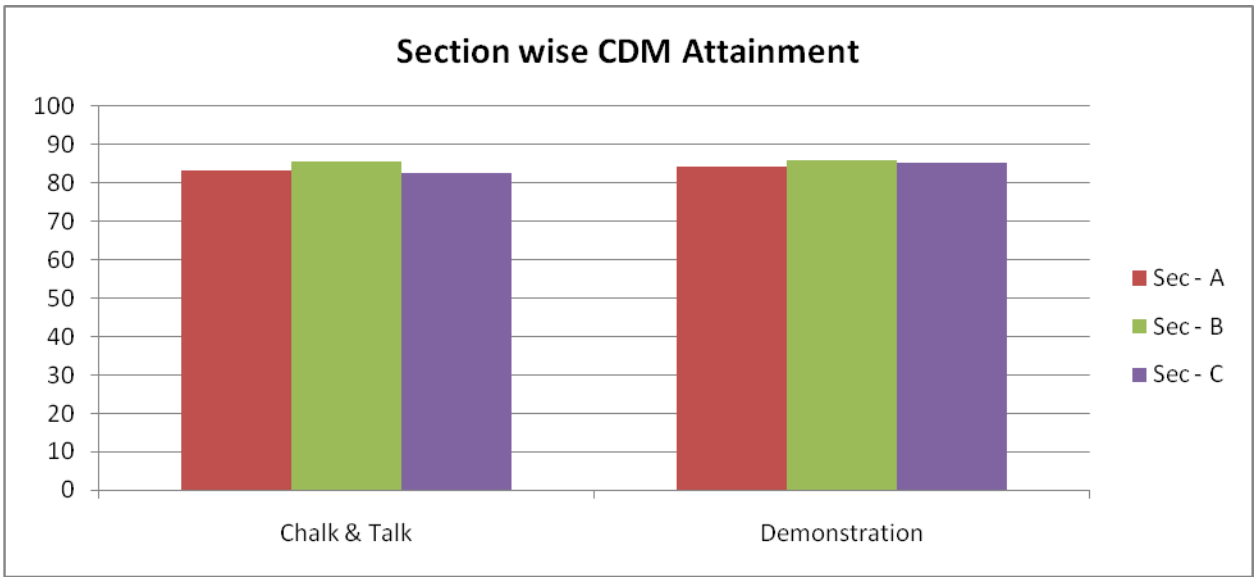
Where

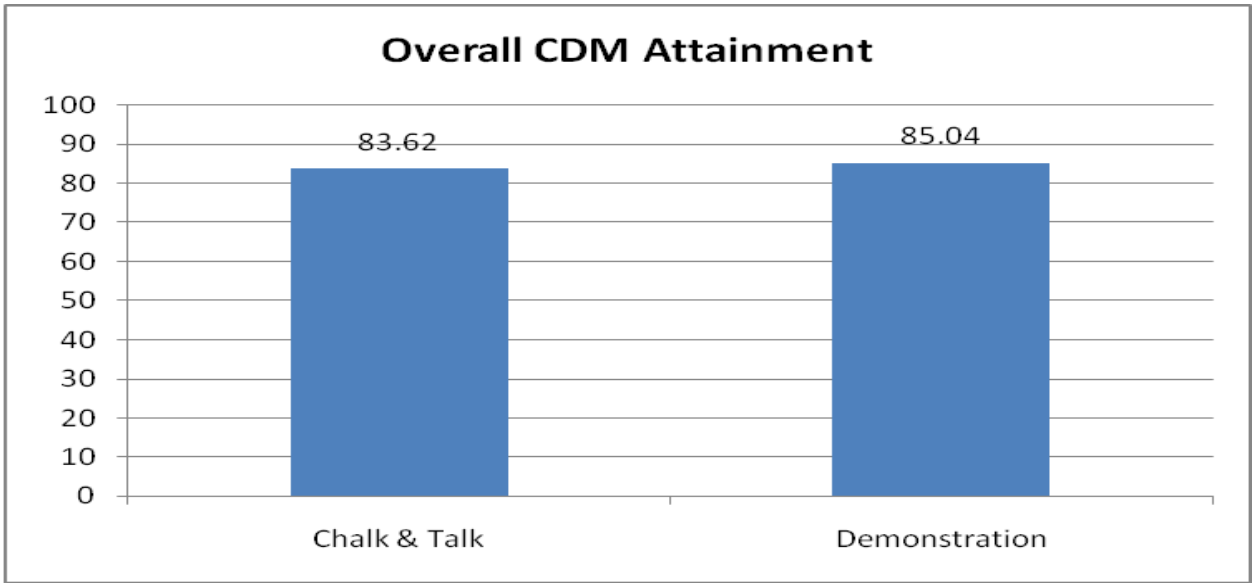
E = Number of students giving Excellent
VG = Number of students giving Very Good
G = Number of students giving Good
S = Number of students giving Satisfactory
P = Number of students giving Poor

N = Total number of students giving the survey

Course Delivery Method (CDM)	CDM Attainment for Sec - A	CDM Attainment for Sec - B	CDM Attainment for Sec - C	Overall Attainment from CDM
Chalk & Talk	83	85.37	82.61	83.62
Demonstrations	84	85.85	85.22	85.04

Note: Attainment in the above table is measured in percentage (%).


Section wise CDM Attainment
100
90
80
70
60
50
40
30
20
10
0
Chalk & Talk
Demonstration
Sec - A
Sec - B
Sec - C


Overall CDM Attainment
100
90
80
70
60
50
40
30
20
10
0
83.62
85.04
Chalk & Talk
Demonstration

COURSE OUTCOME (CO) ATTAINMENT MEASUREMENT

In this course, we have two types of assessment to be carried out. They are as follows,

i. Direct Assessment – Includes Lab Internal Assessment (IA) Test, Record evaluation and Viva
ii. Indirect Assessment – Includes Course End Survey

Thus, here the Continuous Internal Evaluation (CIE) has four components – Lab IA test, Record evaluation, Viva and Course End Survey.

14 a) COURSE OUTCOME ATTAINMENT PROCEDURE FOR DIRECT ASSESSMENT

Direct Assessment Includes Lab Internal Assessment (IA) Test, Record evaluation and Viva. Lab IA test is carried out for a total of 30 marks (Part A – 15 marks and Part B – 15 marks), Record evaluation for 10 marks and Viva for 10 marks.

FORMULA FOR CO ATTAINMENT MEASUREMENT THROUGH Lab IA TEST

Sl. No.	CO #	Formula
1	CO 1	$\frac{M_A \text{ X100}}{N_A \text{ X } 15}$
2	CO 2	
3	CO 3	
4	CO 4	
5	CO 5	$\frac{M_B \text{ X100}}{N_B \text{ X } 15}$

Where,

M_A = Sum of marks scored by each student in Test 1 (Part-A).
N_A = Number of students taking Test 1 (Part-A).
M_B = Sum of marks scored by each student in Test 2 (Part-B).
N_B = Number of students taking Test 2 (Part-B).

FORMULA FOR CO ATTAINMENT MEASUREMENT THROUGH RECORD EVALUATION:

All course outcome attainment through record evaluation is calculated using one common formula and same inputs.

$$\text{Attainment} = \frac{M_R \text{ X } 100}{N \text{ X } 10}$$

Where,

M_R = Sum of marks scored by each student in record evaluation.
N = Number of students.

FORMULA FOR CO ATTAINMENT MEASUREMENT THROUGH VIVA

All course outcome attainment through viva is calculated using one common formula and same inputs.

$$\text{Attainment} = \frac{M_V \text{ X } 100}{N \text{ X } 10}$$

Where,

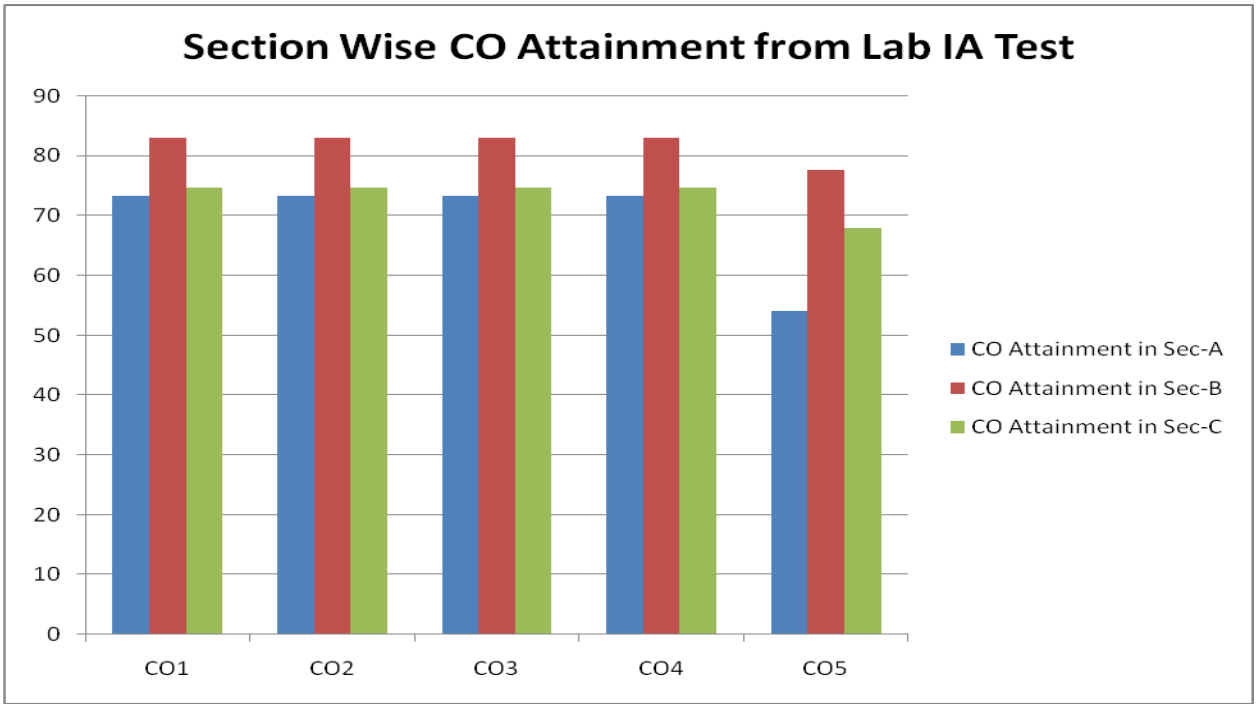
M_V = Sum of marks scored by each student in Viva.

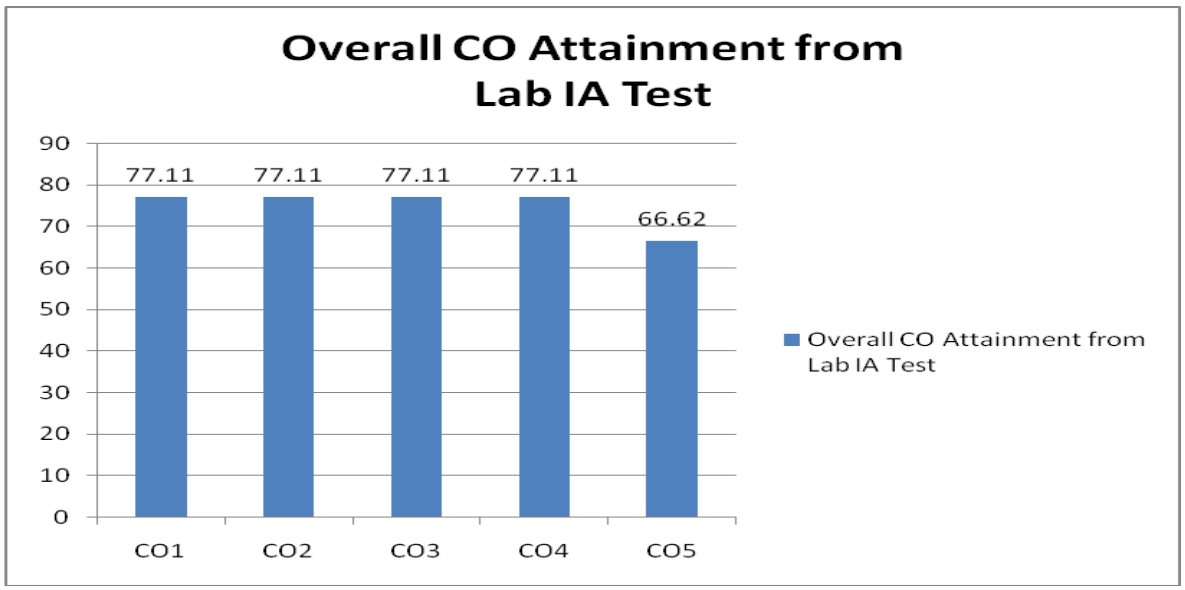
N = Number of students.

14 b) COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH LAB IA TEST

Course Outcome	CO Attainment from Lab IA Test for Sec - A	CO Attainment from Lab IA Test for Sec - B	CO Attainment from Lab IA Test for Sec - C	Overall CO Attainment from Lab IA Test
CO1	73.33	82.99	74.78	77.11
CO2	73.33	82.99	74.78	77.11
CO3	73.33	82.99	74.78	77.11
CO4	73.33	82.99	74.78	77.11
CO5	54.03	77.69	67.97	66.62

Note: CO Attainment in the above table is measured in percentage (%).





14 c) COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH RECORD EVALUATION

Course Outcome	CO Attainment from Record Evaluation for Sec - A	CO Attainment from Record Evaluation for Sec - B	CO Attainment from Record Evaluation for Sec - C	Overall CO Attainment from Record Evaluation
CO1	86.46	84.69	88.26	86.43
CO2	86.46	84.69	88.26	86.43
CO3	86.46	84.69	88.26	86.43
CO4	86.46	84.69	88.26	86.43
CO5	86.46	84.69	88.26	86.43

Note: CO Attainment in the above table is measured in percentage (%).

Graph not required since all CO's have the same attainment level.

14 d) COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH VIVA

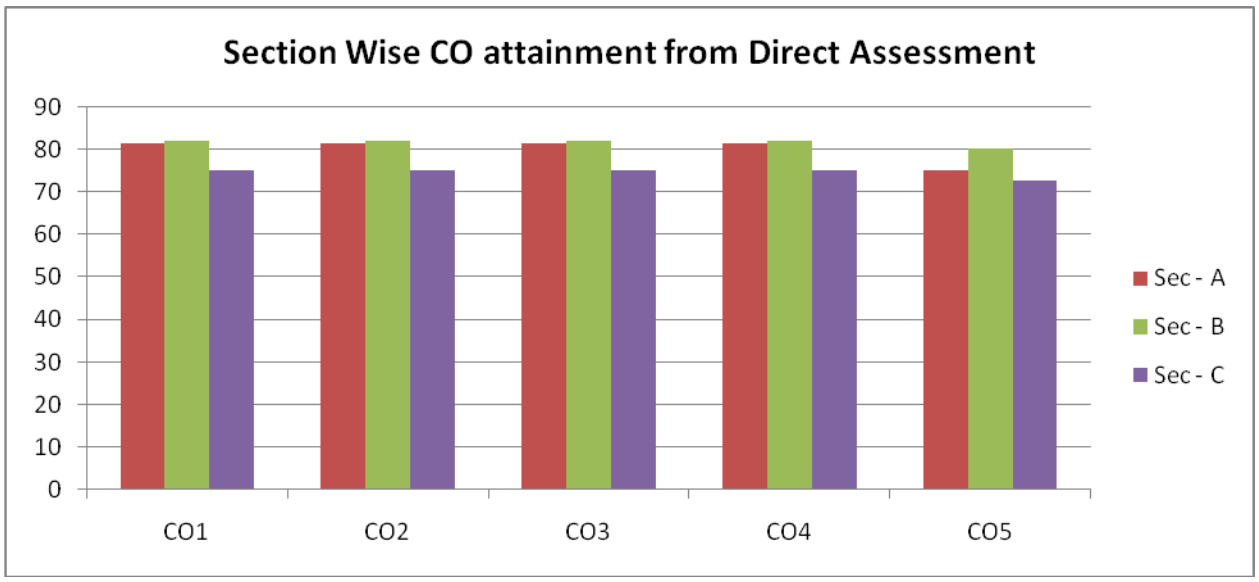
Course Outcome	CO Attainment from Viva for Sec - A	CO Attainment from Viva for Sec - B	CO Attainment from Viva for Sec - C	Overall CO Attainment from Viva
CO1	84.79	78.57	62.17	75.38
CO2	84.79	78.57	62.17	75.38
CO3	84.79	78.57	62.17	75.38
CO4	84.79	78.57	62.17	75.38
CO5	84.79	78.57	62.17	75.38

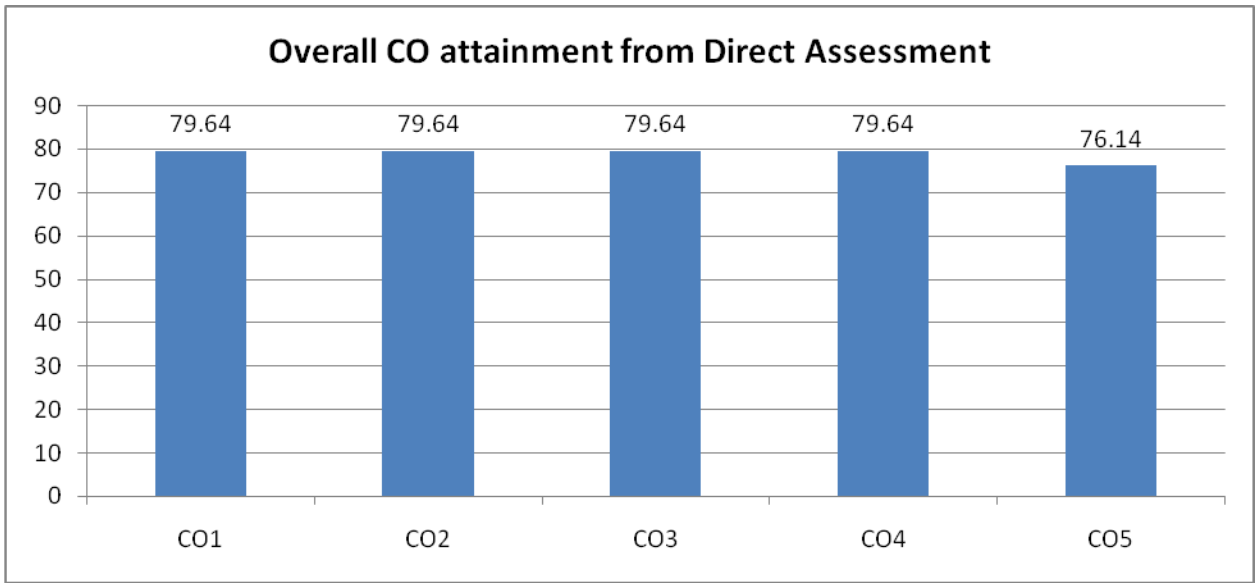
Note: CO Attainment in the above table is measured in percentage (%).

Graph not required since all CO's have the same attainment level.

14 e) COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH DIRECT ASSESSMENT

Course Outcome	CO Attainment from Direct Assessment for Sec - A	CO Attainment from Direct Assessment for Sec - B	CO Attainment from Direct Assessment for Sec - C	Overall CO Attainment from Direct Assessment
CO1	81.53	82.08	75.07	79.64
CO2	81.53	82.08	75.07	79.64
CO3	81.53	82.08	75.07	79.64
CO4	81.53	82.08	75.07	79.64
CO5	75.09	80.32	72.8	76.14





15 a) COURSE OUTCOME ATTAINMENT PROCEDURE EXPLANATION FOR IN-DIRECT ASSESSMENT (COURSE END SURVEY)

The formula to measure PO attainment level through direct assessment methods is:

$$\text{CO Attainment} = \frac{((5 * E) + (4 * VG) + (3* G) + (2* S) +(1 * P)) \text{ X}100}{(N \text{ X } 5)}$$

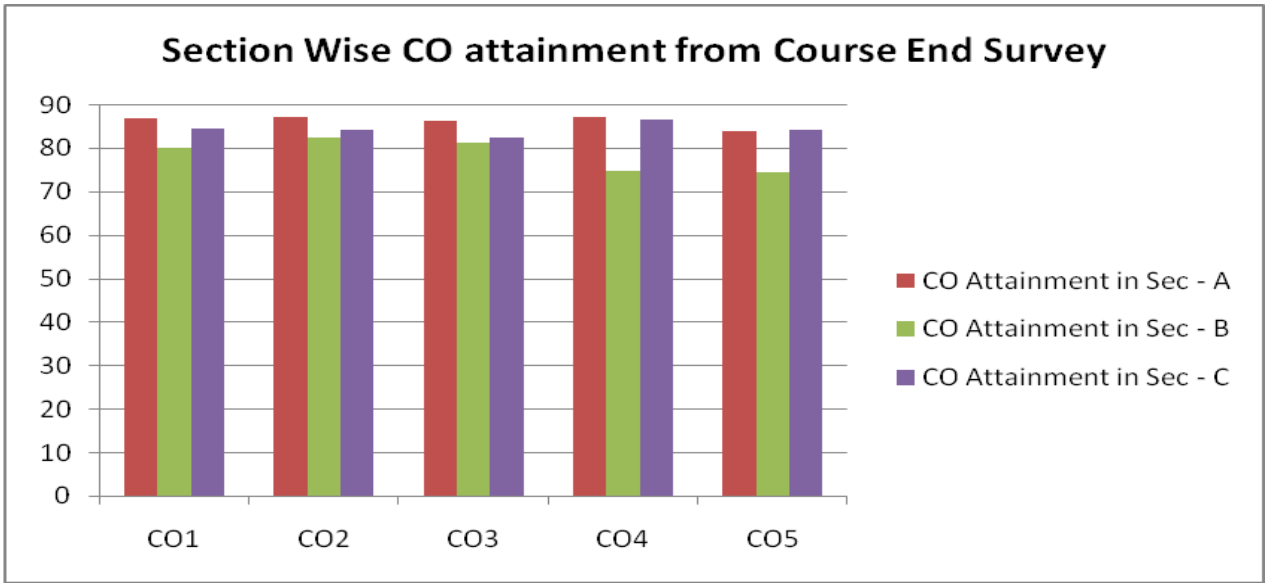
Where

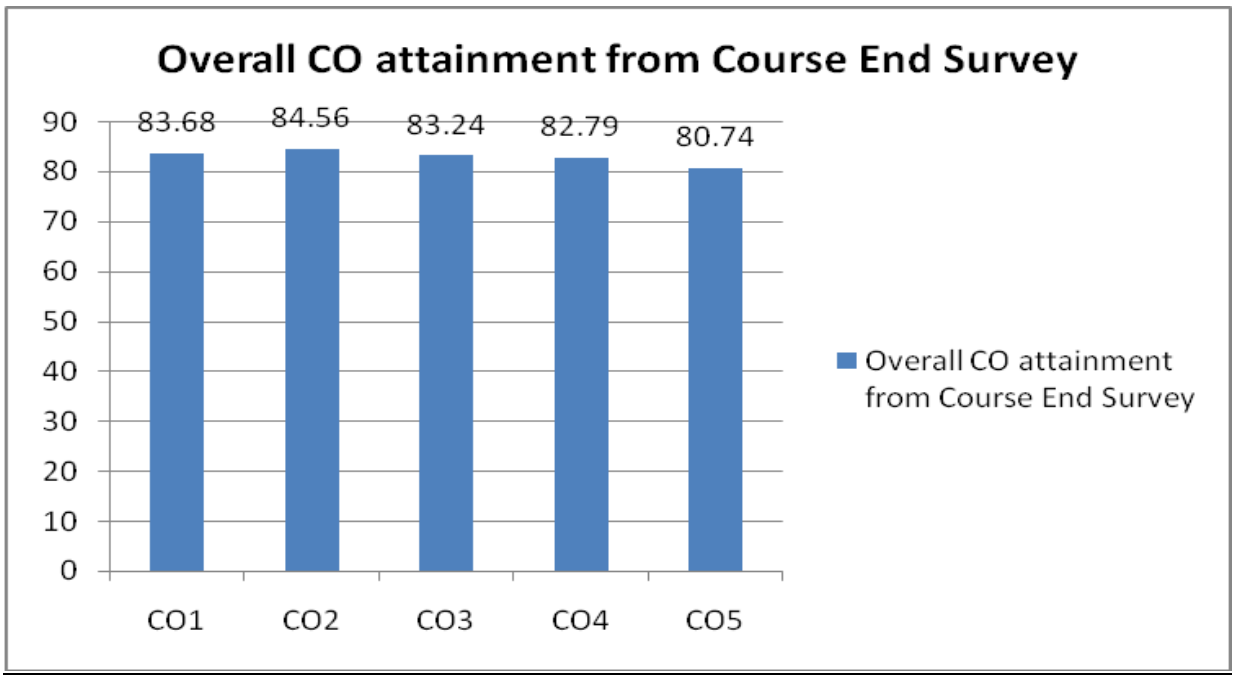
E = Number of students giving Excellent
VG = Number of students giving Very Good
G = Number of students giving Good
S = Number of students giving Satisfactory
P = Number of students giving Poor
N = Total number of students giving the survey

15 b) COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH IN-DIRECT ASSESSMENT (COURSE END SURVEY)

Course Outcome	CO Attainment from CES for Sec - A	CO Attainment from CES for Sec - B	CO Attainment from CES for Sec - C	Overall CO Attainment from CES
CO1	86.83	80	84.58	83.68
CO2	87.32	82.55	84.17	84.56
CO3	86.34	81.28	82.5	83.24
CO4	87.32	74.89	86.67	82.79
CO5	83.9	74.47	84.17	80.74

Note: CO Attainment in the above table is measured in percentage (%).





16 a) OVERALL COURSE OUTCOME ATTAINMENT PROCEDURE THROUGH DIRECT AND IN-DIRECT ASSESSMENT

RUBRIC:

Sl. No.	CIE Components	Weightage
1	Direct Assessment	0.7
2	In-Direct Assessment	0.3

FORMULA:

$CO_i = (0.7 \text{ X } CO_i\_A_D) + (0.3 \text{ X } CO_i\_A_I)$

Where

CO_i = Course Outcome i (i takes the values 1, 2, 3, 4, 5)

$CO_i\_A_D$ = Attainment of Course Outcome i from Direct Assessment

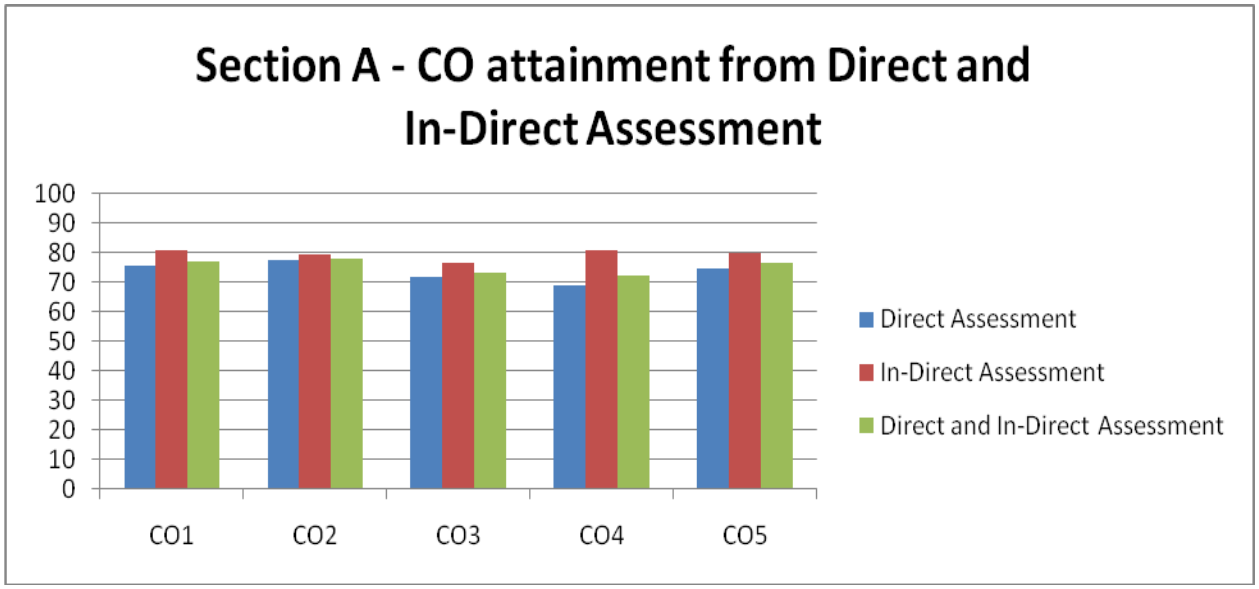
$CO_i\_A_I$ = Attainment of Course Outcome i from In-Direct Assessment

16 b) OVERALL COURSE OUTCOME ATTAINMENT MEASUREMENT THROUGH DIRECT AND IN-DIRECT ASSESSMENT

SECTION A:

Course Outcome	Section A		
	CO Attainment from Direct Assessment	CO Attainment from In-Direct Assessment	Overall CO Attainment from Direct and In-Direct Assessment
CO1	81.53	86.83	83.12
CO2	81.53	87.32	83.27
CO3	81.53	86.34	82.97
CO4	81.53	87.32	83.27
CO5	75.09	83.9	77.73

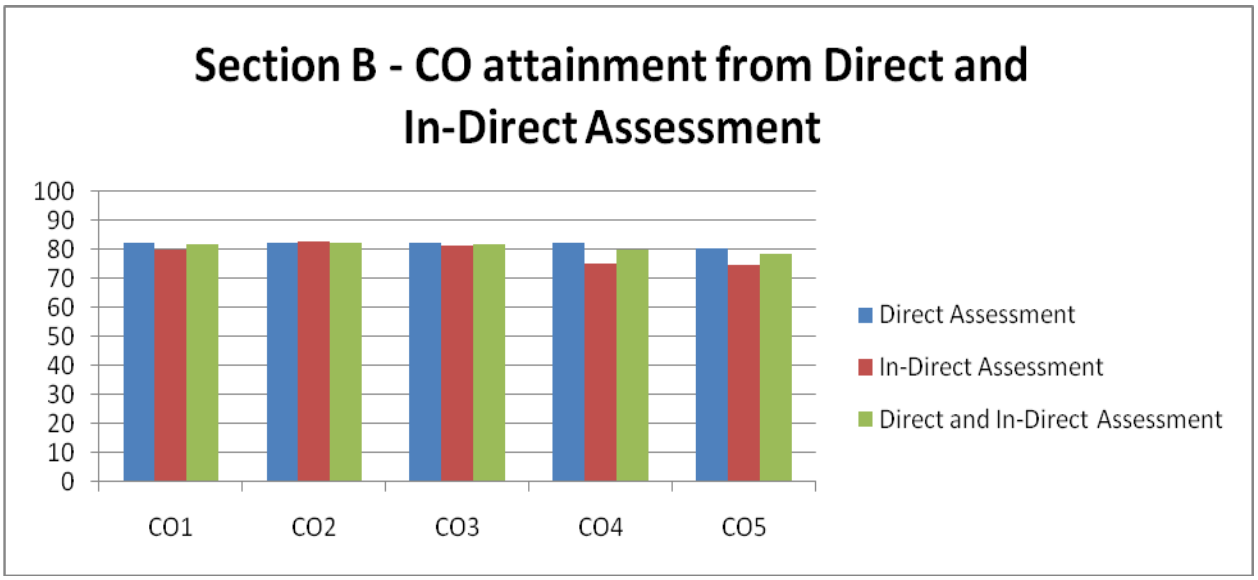
Note: CO Attainment in the above table is measured in percentage (%).



SECTION B:

Course Outcome	Section B		
	CO Attainment from Direct Assessment	CO Attainment from In-Direct Assessment	Overall CO Attainment from Direct and In-Direct Assessment
CO1	82.08	80	81.46
CO2	82.08	82.55	82.22
CO3	82.08	81.28	81.84
CO4	82.08	74.89	79.92
CO5	80.32	74.47	78.57

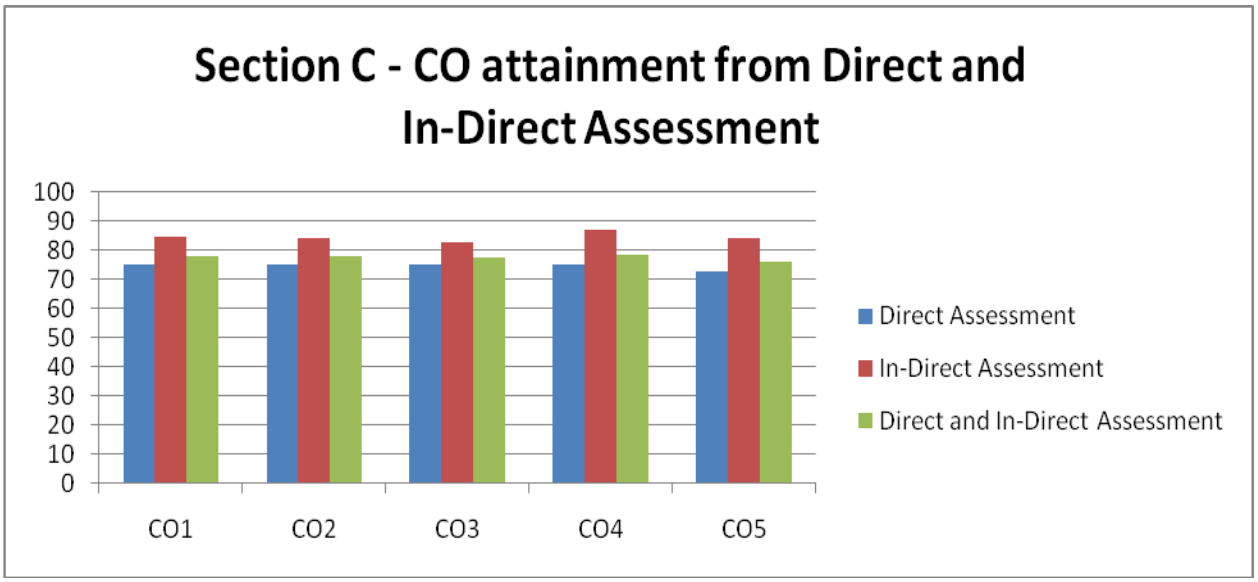
Note: CO Attainment in the above table is measured in percentage (%).



SECTION C:

Course Outcome	Section C		
	CO Attainment from Direct Assessment	CO Attainment from In-Direct Assessment	Overall CO Attainment from Direct and In-Direct Assessment
CO1	75.07	84.58	77.92
CO2	75.07	84.17	77.8
CO3	75.07	82.5	77.3
CO4	75.07	86.67	78.55
CO5	72.8	84.17	76.21

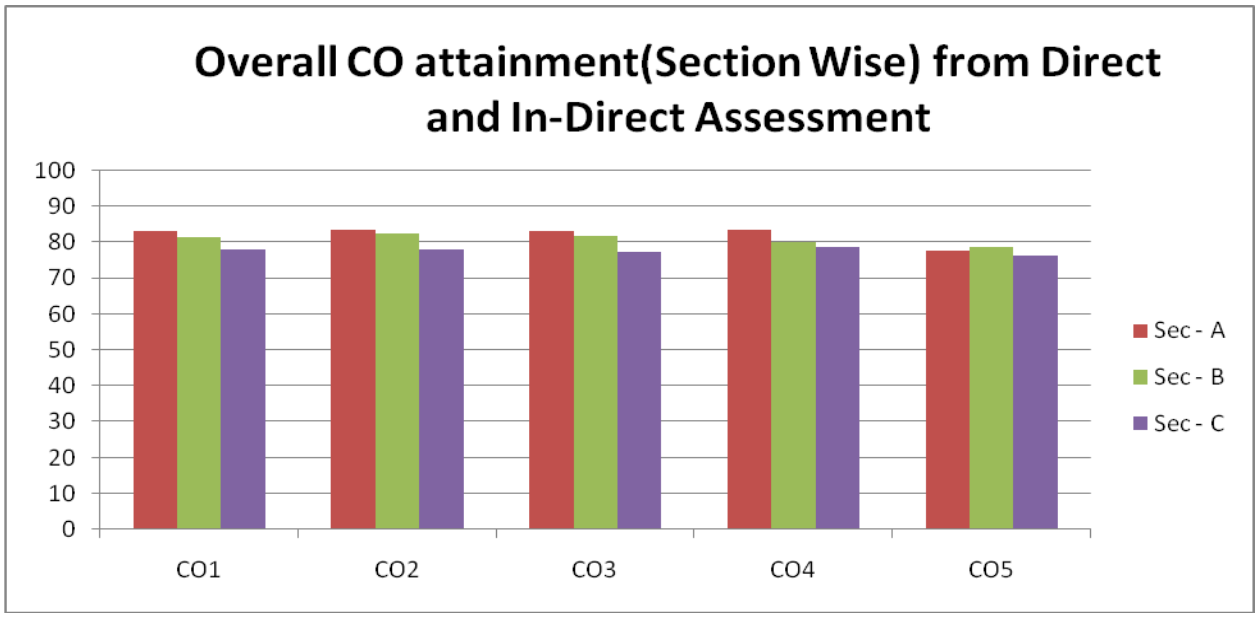
Note: CO Attainment in the above table is measured in percentage (%).

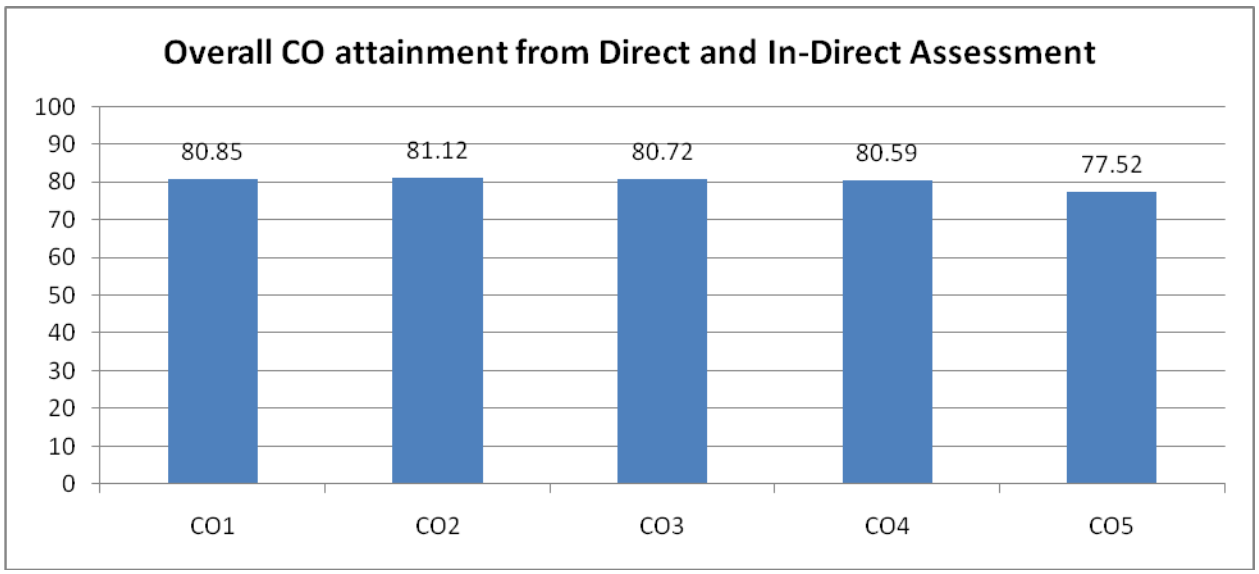


OVERALL COURSE OUTCOME ATTAINMENT:

Course Outcome	Overall CO Attainment from Direct and In-Direct Assessment for Sec - A	Overall CO Attainment from Direct and In-Direct Assessment for Sec - B	Overall CO Attainment from Direct and In-Direct Assessment for Sec - C	Overall CO Attainment from Direct and In-Direct Assessment
CO1	83.12	81.46	77.92	80.85
CO2	83.27	82.22	77.8	81.12
CO3	82.97	81.84	77.3	80.72
CO4	83.27	79.92	78.55	80.59
CO5	77.73	78.57	76.21	77.52

Note: CO Attainment in the above table is measured in percentage (%).





17) OVERALL COURSE OUTCOME (CO) ATTAINMENT THROUGH DIRECT AND INDIRECT ASSESSMENT GAP ANALYSIS – WHY NOT 100%

Course Outcome	CO1	CO2	CO3	CO4	CO5
CO Attainment	80.85	81.12	80.72	80.59	77.52

Note: CO Attainment in the above table is measured in percentage (%).

This course is offered to 6th semester students as a core laboratory subject. The course have been satisfactorily completed with all course outcome attainment of above 75% but still failed to achieve 100%. More effective teaching methodologies will be adopted next time to increase the attainment and fill the gap.
Reducing the gap is a significant challenge. The various strategies to reduce the gap are as follows:

- Focus on improving teaching and learning methods - experimenting the new teaching methodologies to make the student learn better.
- Collaborative and cooperative learning – providing individual attention to the dull students as far as possible.
- Peer involvement in learning – helping the poor students in their subjects by taking the help of the bright students of the class and following a mentor system.

18) OVERALL COURSE OUTCOME (CO) ATTAINMENT THROUGH DIRECT AND INDIRECT ASSESSMENT GAP ANALYSIS – COMPARISON WITH LAST YEAR

Course Outcome	CO Attainment for the academic year 2013-2014	CO Attainment for the academic year 2012-2013
CO1	80.85	Attainment through CIE – Micro analysis not done.
CO2	81.12	
CO3	80.72	
CO4	80.59	
CO5	77.52	

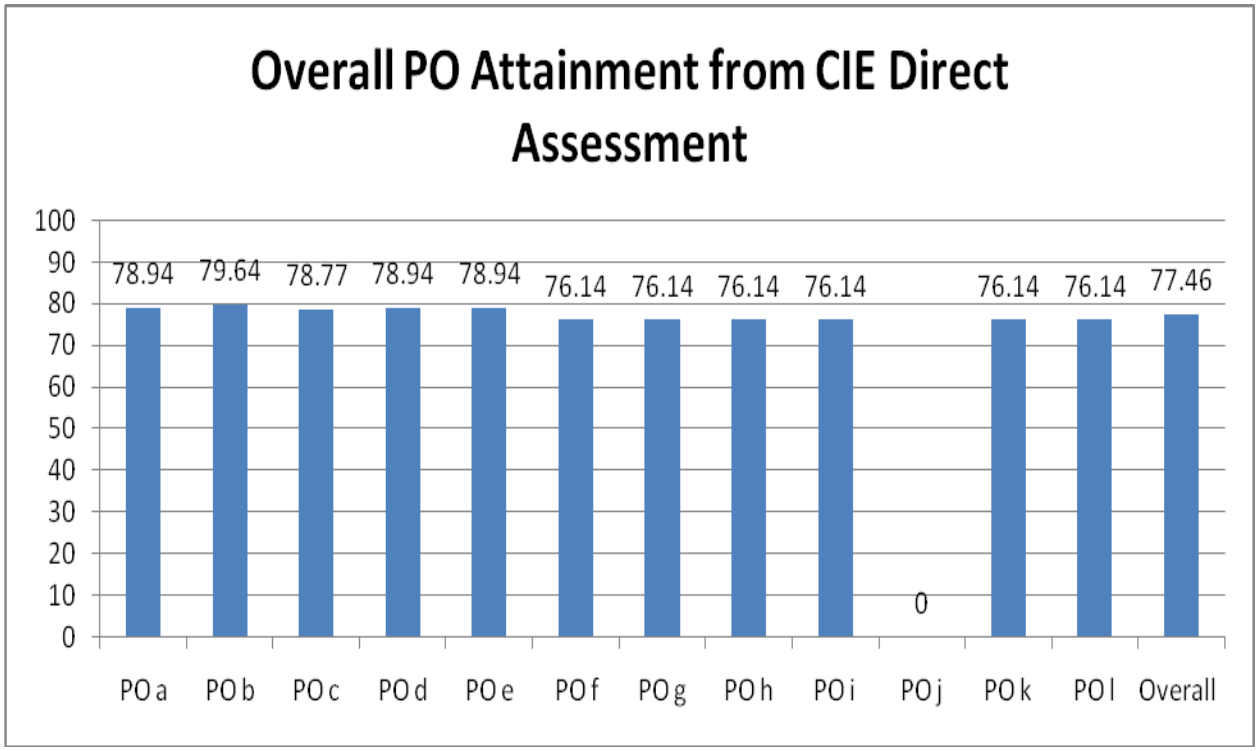
Note: CO Attainment in the above table is measured in percentage (%).

This course comes with an updated syllabus and course outcomes when compared with the similar course offered in the previous academic year. The attainment of the similar course in the previous term is not calculated. However, attainment in the current term can be further increased. The attainment also depends on the students urge to learn the subject and following the instructions of the teacher who is striving to improve the performance of the individual student and to increase the course outcome by bridging the gap. From the analysis it is seen that the attainment needs to be improved a lot and a corrective measure as mentioned in the Section 17 can be followed to bridge the gap and achieve 100 % attainment in this course.

PO ATTAINMENT MEASUREMENT

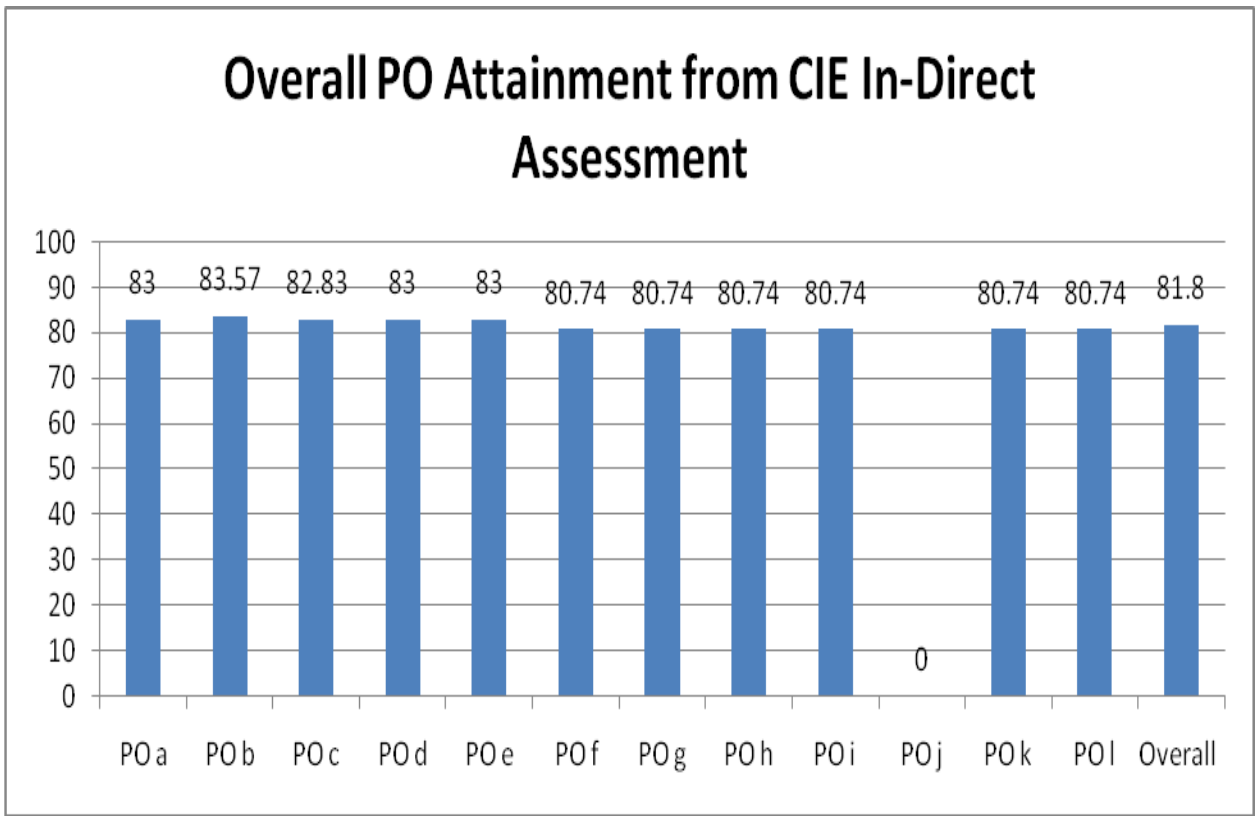
19) PO ATTAINMENT THROUGH CIE (DIRECT ASSESSMENT MICRO-ANALYSIS)

Program Outcomes	Course Outcomes Attainment Values					Average
	CO 1	CO 2	CO 3	CO 4	CO 5	
PO a	79.64	79.64	79.64	79.64	76.14	78.94
PO b	79.64	79.64	79.64	79.64		79.64
PO c		79.64	79.64	79.64	76.14	78.77
PO d	79.64	79.64	79.64	79.64	76.14	78.94
PO e	79.64	79.64	79.64	79.64	76.14	78.94
PO f					76.14	76.14
PO g					76.14	76.14
PO h					76.14	76.14
PO i					76.14	76.14
PO j						0
PO k					76.14	76.14
PO l					76.14	76.14
Overall PO Attainment						77.46



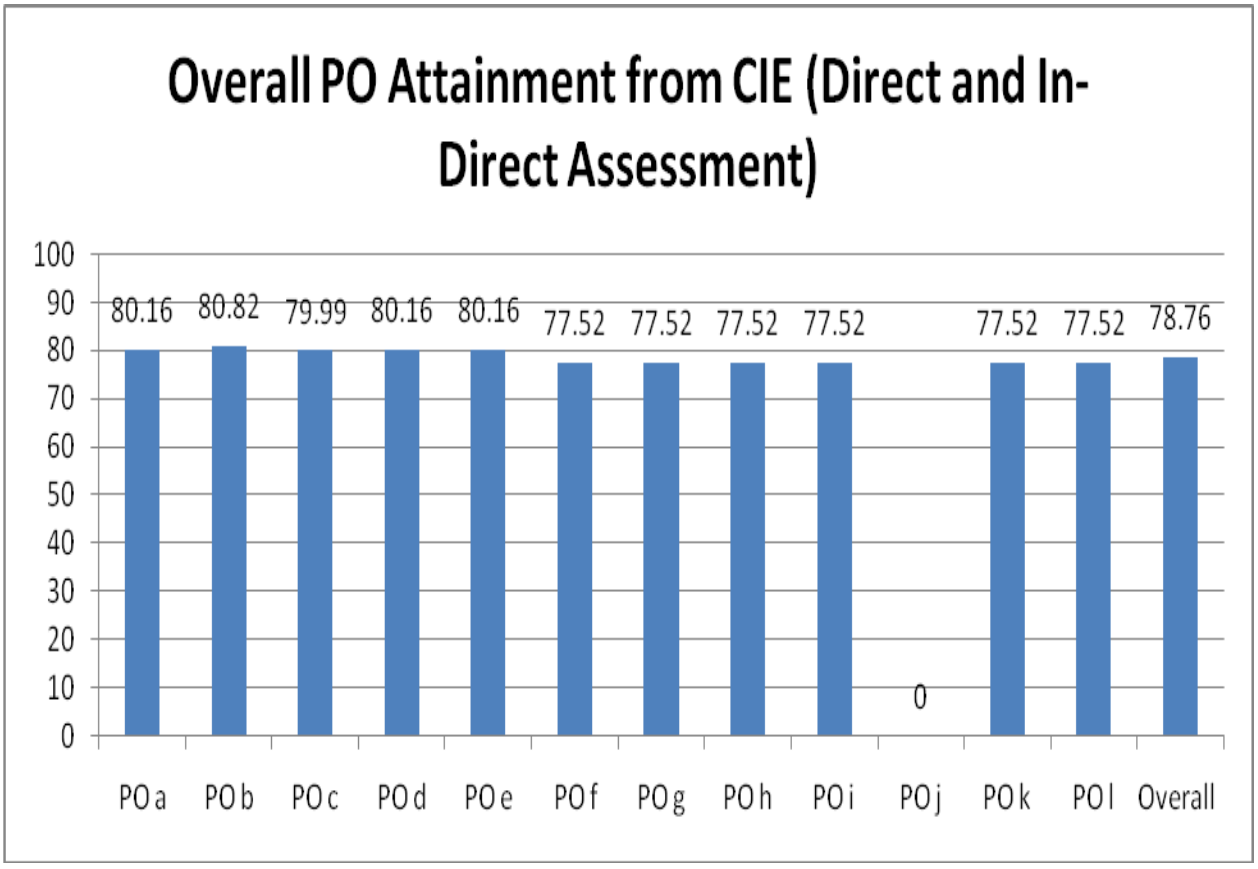
20) PO ATTAINMENT THROUGH CIE (IN-DIRECT ASSESSMENT ANALYSIS - COURSE END SURVEY)

Program Outcomes	Course Outcomes Attainment Values					Average
	CO 1	CO 2	CO 3	CO 4	CO 5	
PO a	83.68	84.56	83.24	82.79	80.74	83
PO b	83.68	84.56	83.24	82.79		83.57
PO c		84.56	83.24	82.79	80.74	82.83
PO d	83.68	84.56	83.24	82.79	80.74	83
PO e	83.68	84.56	83.24	82.79	80.74	83
PO f					80.74	80.74
PO g					80.74	80.74
PO h					80.74	80.74
PO i					80.74	80.74
PO j						0
PO k					80.74	80.74
PO l					80.74	80.74
Overall PO Attainment						81.8



21) OVERALL PO ATTAINMENT THROUGH CIE (DIRECT AND IN-DIRECT ASSESSMENT)

Program Outcomes	Course Outcomes Attainment Values					Average
	CO 1	CO 2	CO 3	CO 4	CO 5	
PO a	80.85	81.12	80.72	80.59	77.52	80.16
PO b	80.85	81.12	80.72	80.59		80.82
PO c		81.12	80.72	80.59	77.52	79.99
PO d	80.85	81.12	80.72	80.59	77.52	80.16
PO e	80.85	81.12	80.72	80.59	77.52	80.16
PO f					77.52	77.52
PO g					77.52	77.52
PO h					77.52	77.52
PO i					77.52	77.52
PO j						0
PO k					77.52	77.52
PO l					77.52	77.52
Overall PO Attainment						78.76



22) SEE OVERALL RESULT ANALYSIS (ALL THREE SECTIONS)

Grade	S	A	B	C	D	E	F
No. of students scoring the grade	22	57	45	16	0	0	0
Weight-age	0.35	0.3	0.15	0.1	0.05	0.05	0
Overall attainment	**67.65 %**						

23) OVERALL PO ATTAINMENT THROUGH SEE DIRECT ASSESSMENT – MACRO ANALYSIS

FORMULA:

The formula to measure PO attainment level through direct assessment methods is:

$$\text{Attainment} = \frac{(0.35 \text{ X } Ns)+(0.3 \text{ X } Na)+(0.15 \text{ X } Nb)+(0.1 \text{ X } Nc)+(0.05 \text{ X } Nd)+(0.05 \text{ X } Ne)}{\text{Total X } 0.35}$$

Where

Ns = Number of Students scoring 'S' grade
Na = Number of Students scoring 'A' grade
Nb = Number of Students scoring 'B' grade
Nc = Number of Students scoring 'C' grade
Nd = Number of Students scoring 'D' grade
Ne = Number of Students scoring 'E' grade
Total = Total number of students who took up the exam. This would exclude Not Eligible students and students who withdrew the course

RANGE OF MARKS AND RESPECTIVE GRADES:

Range of Marks	**Grade**
Marks >= 90%	S
75% <= Marks < 90%	A
60% <= Marks < 75%	B
50% <= Marks < 60%	C
45% <= Marks < 50%	D
40% <= Marks < 45%	E
Marks < 40%	F

PO's SATISFIED BY THE COURSE:

PO a	PO b	PO c	PO d	PO e	PO f	PO g	PO h	PO i	PO j	PO k	PO l
X	X	X	X	X	X	X	X	X		X	X

CO ATTAINMENT THROUGH SEE DIRECT ASSESSMENT:

Course Outcome	CO Attainment from SEE Direct Assessment
CO1	67.65
CO2	67.65
CO3	67.65
CO4	67.65
CO5	67.65

Note: CO Attainment in the above table is measured in percentage (%).
Graph not required since all CO's have the same attainment level.

PO ATTAINMENT THROUGH SEE DIRECT ASSESSMENT:

Program Outcomes	Course Outcomes Attainment Values					Average
	CO 1	CO 2	CO 3	CO 4	CO 5	
PO a	67.65	67.65	67.65	67.65	67.65	67.65
PO b	67.65	67.65	67.65	67.65		67.65
PO c		67.65	67.65	67.65	67.65	67.65
PO d	67.65	67.65	67.65	67.65	67.65	67.65
PO e	67.65	67.65	67.65	67.65	67.65	67.65
PO f					67.65	67.65
PO g					67.65	67.65
PO h					67.65	67.65
PO i					67.65	67.65
PO j						0
PO k					67.65	67.65
PO l					67.65	67.65
Overall PO Attainment						67.65

Note: Attainment in the above table is measured in percentage (%).
Graph not required since all CO's have the same attainment level.

24) OVERALL PO ATTAINMENT THROUGH SEE DIRECT AND IN-DIRECT ASSESSMENT: MACRO ANALYSIS

The same Course End Survey taken as a part of CIE In-Direct Assessment is also considered for the SEE In-Direct Assessment.

RUBRIC:

Sl. No.	CIE Components	Weightage
1	Direct Assessment	0.7
2	In-Direct Assessment	0.3

FORMULA:

$$CO_i = (0.7 \text{ X } CO_i\_A_D) + (0.3 \text{ X } CO_i\_A_I)$$

Where

CO_i = Course Outcome i (i takes the values 1, 2, 3, 4, 5)

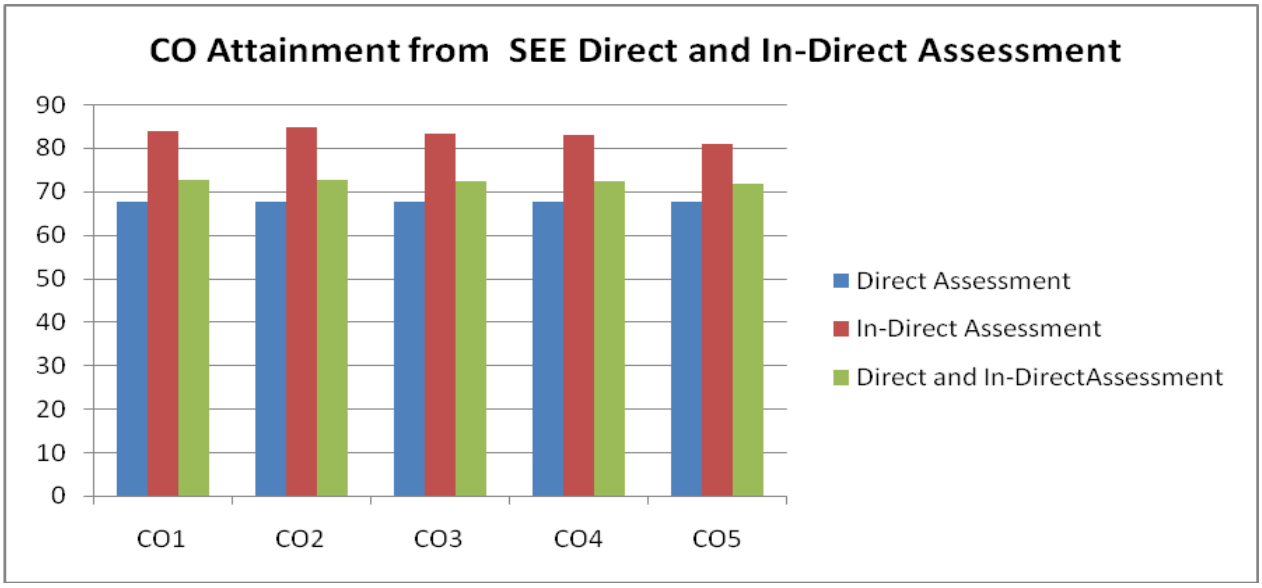
$CO_i\_A_D$ = Attainment of Course Outcome i from SEE Direct Assessment

$CO_i\_A_I$ = Attainment of Course Outcome i from SEE In-Direct Assessment

CO ATTAINMENT THROUGH SEE DIRECT AND IN-DIRECT ASSESSMENT:

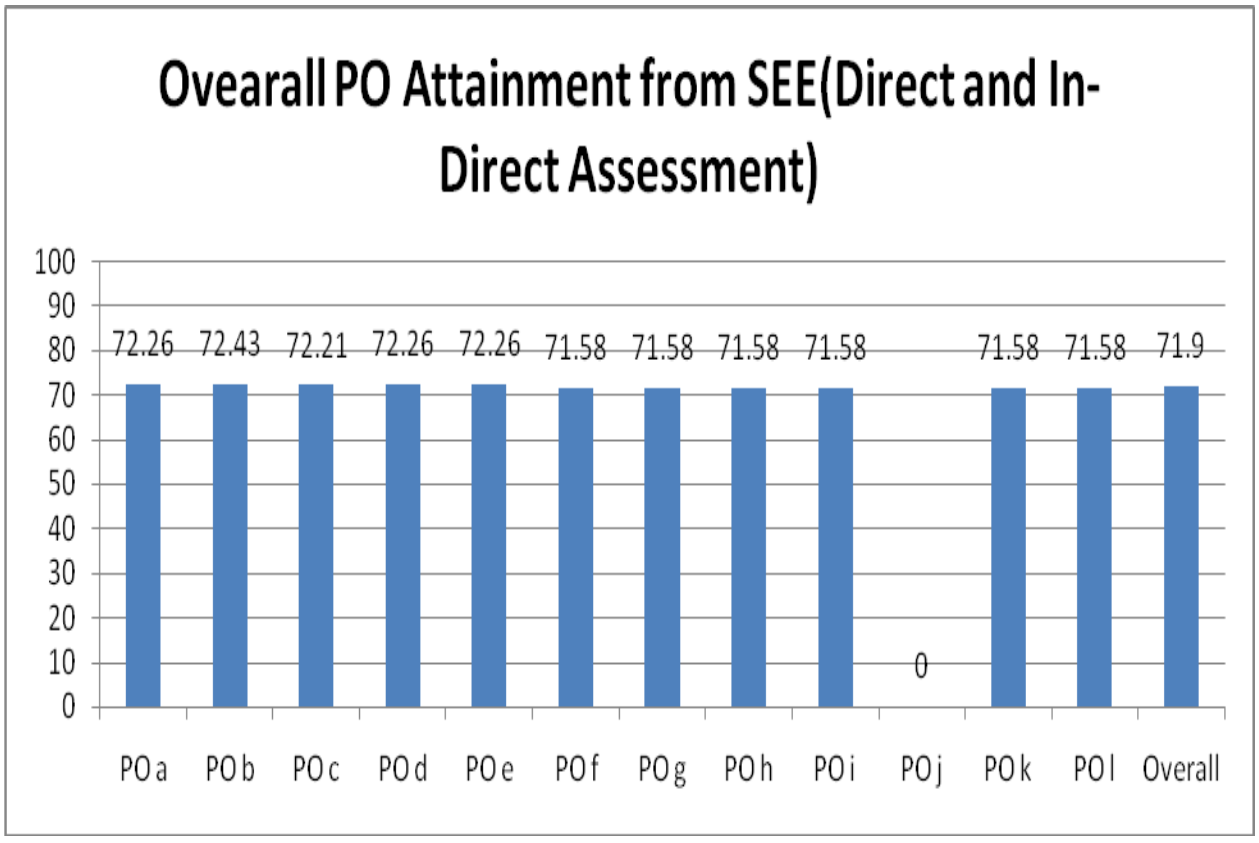
Course Outcome	CO Attainment from SEE Direct Assessment	CO Attainment from SEE In-Direct Assessment	Overall CO Attainment from SEE Direct and In-Direct Assessment
CO1	67.65	83.68	72.46
CO2	67.65	84.56	72.72
CO3	67.65	83.24	72.33
CO4	67.65	82.79	72.19
CO5	67.65	80.74	71.58

Note: CO Attainment in the above table is measured in percentage (%).



PO ATTAINMENT THROUGH SEE DIRECT AND IN-DIRECT ASSESSMENT:

Program Outcomes	Course Outcomes Attainment Values					Average
	CO 1	CO 2	CO 3	CO 4	CO 5	
PO a	72.46	72.72	72.33	72.19	71.58	72.26
PO b	72.46	72.72	72.33	72.19		72.43
PO c		72.72	72.33	72.19	71.58	72.21
PO d	72.46	72.72	72.33	72.19	71.58	72.26
PO e	72.46	72.72	72.33	72.19	71.58	72.26
PO f					71.58	71.58
PO g					71.58	71.58
PO h					71.58	71.58
PO i					71.58	71.58
PO j						0
PO k					71.58	71.58
PO l					71.58	71.58
Overall PO Attainment						71.9



25) OVERALL PO ATTAINMENT THROUGH SEE - GAP ANALYSIS

EXPLANATION

Program Outcomes	PO a	PO b	PO c	PO d	PO e	PO f	PO g	PO h	PO i	PO j	PO k	PO l
Attainment	72.26	72.43	72.21	72.26	72.26	71.58	71.58	71.58	71.58	0	71.58	71.58

Note: Attainment in the above table is measured in percentage (%).