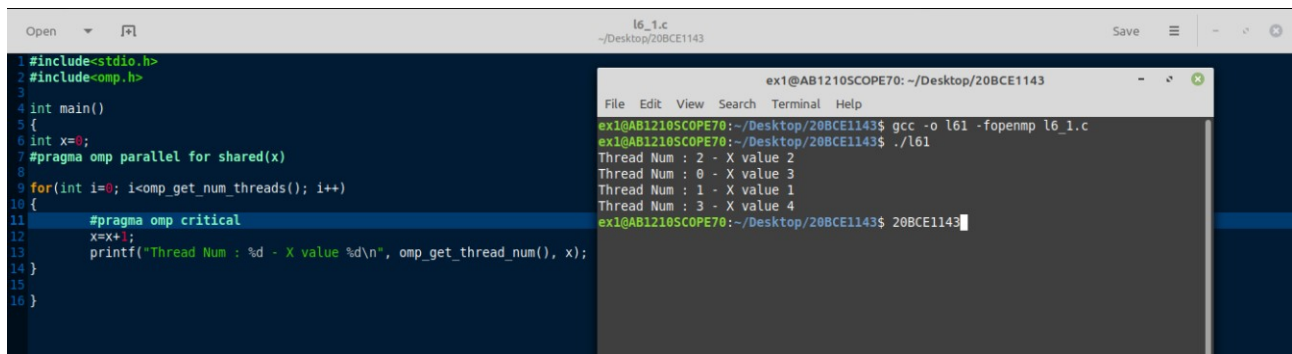


# PDC Lab 6

Advait Deochakke

20BCE1143

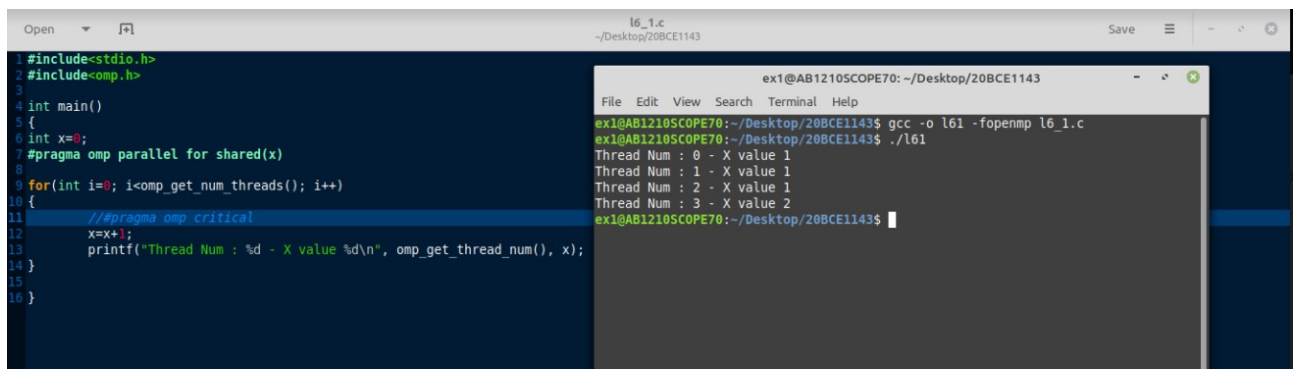
Sample critical without parallel program



The screenshot shows a code editor with a C program named `l6_1.c`. The program includes `<stdio.h>` and `<omp.h>`. It has a `main` function that initializes `x=0` and enters a parallel region using `#pragma omp parallel for shared(x)`. Inside the parallel region, there is a loop from `i=0` to `omp_get_num_threads()-1`. Each iteration contains a critical section marked by `#pragma omp critical` where `x` is incremented and a message is printed. The terminal output shows four threads (0, 1, 2, 3) each printing their thread number and the value of `x` after incrementing. The values of `x` are 2, 3, 1, and 4 respectively, indicating that the critical section successfully prevented race conditions.

```
1 #include<stdio.h>
2 #include<omp.h>
3
4 int main()
5 {
6     int x=0;
7     #pragma omp parallel for shared(x)
8     for(int i=0; i<omp_get_num_threads(); i++)
9     {
10         #pragma omp critical
11         x=x+1;
12         printf("Thread Num : %d - X value %d\n", omp_get_thread_num(), x);
13     }
14 }
15
16 }
```

```
ex1@AB1210SCOPE70: ~/Desktop/20BCE1143
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ gcc -o l61 -fopenmp l6_1.c
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ ./l61
Thread Num : 2 - X value 2
Thread Num : 0 - X value 3
Thread Num : 1 - X value 1
Thread Num : 3 - X value 4
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ 20BCE1143
```



The screenshot shows the same code editor and terminal as above, but with a different output. The program is the same, but the terminal output shows different values for `x` after each thread's execution. This suggests a different run or a different configuration.

```
1 #include<stdio.h>
2 #include<omp.h>
3
4 int main()
5 {
6     int x=0;
7     #pragma omp parallel for shared(x)
8     for(int i=0; i<omp_get_num_threads(); i++)
9     {
10         //pragma omp critical
11         x=x+1;
12         printf("Thread Num : %d - X value %d\n", omp_get_thread_num(), x);
13     }
14 }
15
16 }
```

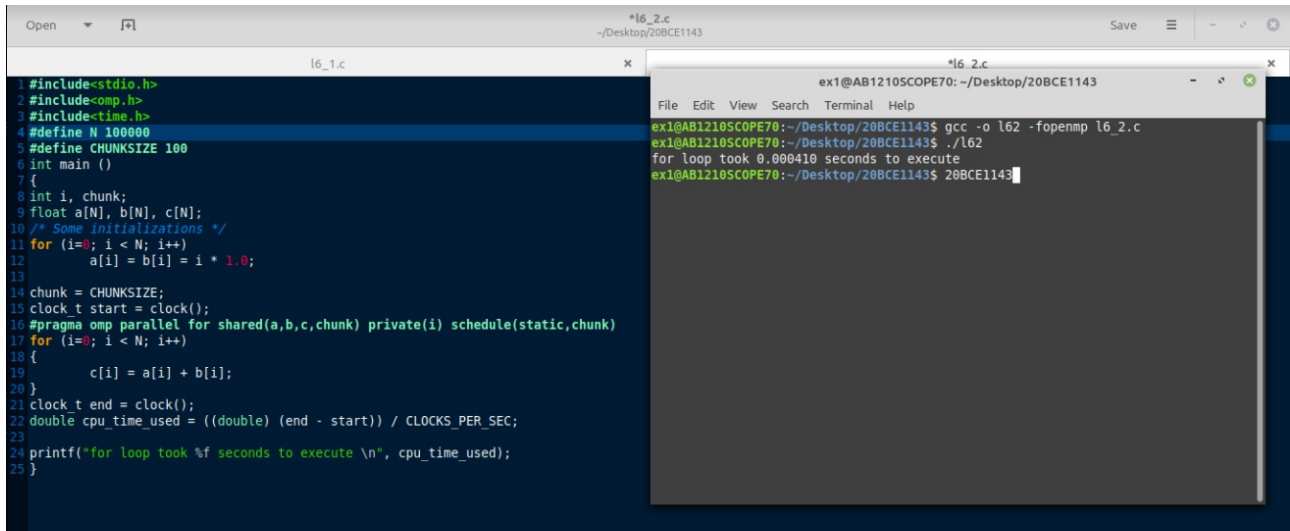
```
ex1@AB1210SCOPE70: ~/Desktop/20BCE1143
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ gcc -o l61 -fopenmp l6_1.c
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ ./l61
Thread Num : 0 - X value 1
Thread Num : 1 - X value 1
Thread Num : 2 - X value 1
Thread Num : 3 - X value 2
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$
```

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

```
int main()
{
    int x=0;
    #pragma omp parallel for shared(x)
```

```
for(int i=0; i<omp_get_num_threads(); i++)
{
    #pragma omp critical
    x=x+1;
    printf("Thread Num : %d - X value %d\n", omp_get_thread_num(), x);
}
}
```

## Sample static schedule program



The screenshot shows a code editor with two tabs: `l6_1.c` and `*l6_2.c`. The `l6_1.c` tab is active and contains the following code:

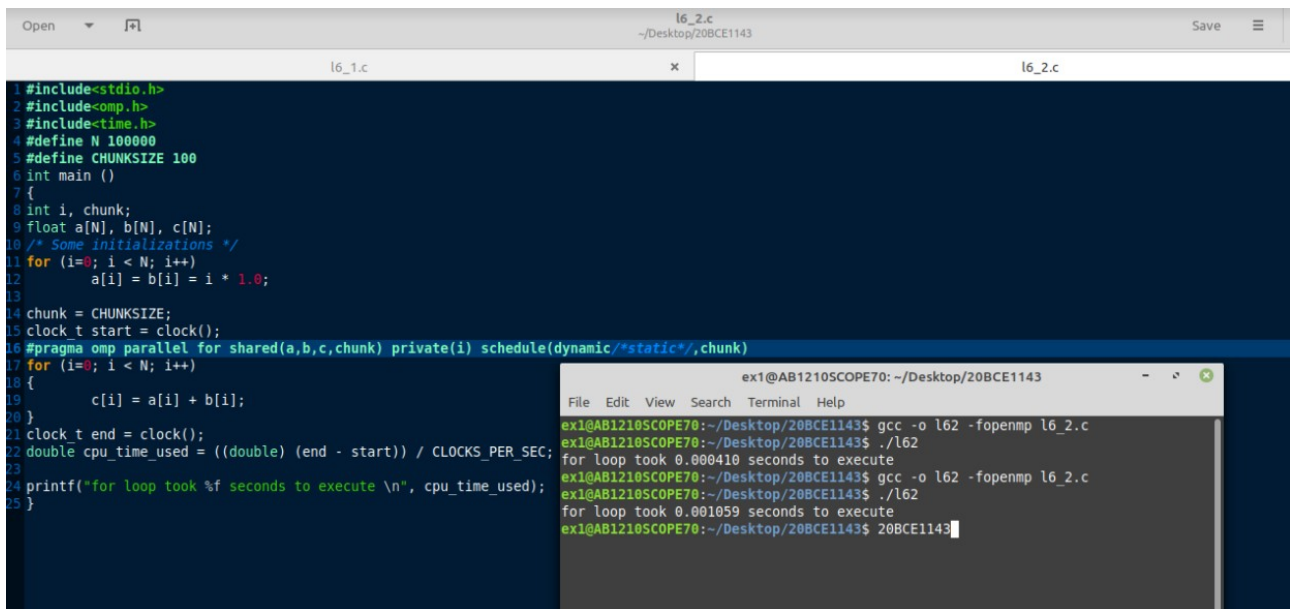
```
1 #include<stdio.h>
2 #include<omp.h>
3 #include<time.h>
4 #define N 100000
5 #define CHUNKSIZE 100
6 int main ()
7 {
8     int i, chunk;
9     float a[N], b[N], c[N];
10    /* Some initializations */
11    for (i=0; i < N; i++)
12        a[i] = b[i] = i * 1.0;
13
14    chunk = CHUNKSIZE;
15    clock_t start = clock();
16    #pragma omp parallel for shared(a,b,c,chunk) private(i) schedule(static,chunk)
17    for (i=0; i < N; i++)
18    {
19        c[i] = a[i] + b[i];
20    }
21    clock_t end = clock();
22    double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
23
24    printf("for loop took %f seconds to execute \n", cpu_time_used);
25 }
```

The `*l6_2.c` tab is also visible and contains the same code. A terminal window is open in the foreground, showing the execution of the program:

```
ex1@AB1210SCOPE70: ~/Desktop/20BCE1143
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ gcc -o l62 -fopenmp l6_2.c
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ ./l62
for loop took 0.000410 seconds to execute
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ 20BCE1143
```

(Code combined with next 4 questions)

## Sample dynamic schedule program



The screenshot shows a code editor with two tabs: `l6_1.c` and `*l6_2.c`. The `l6_1.c` tab is active and contains the following code:

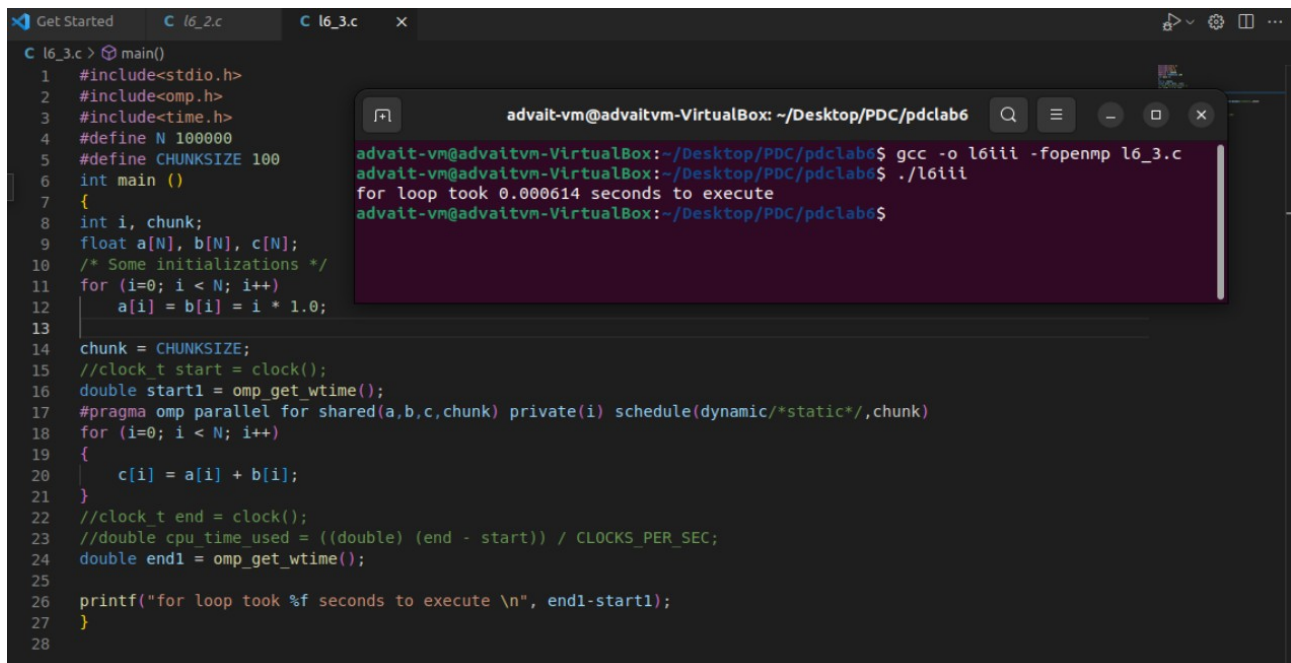
```
1 #include<stdio.h>
2 #include<omp.h>
3 #include<time.h>
4 #define N 100000
5 #define CHUNKSIZE 100
6 int main ()
7 {
8     int i, chunk;
9     float a[N], b[N], c[N];
10    /* Some initializations */
11    for (i=0; i < N; i++)
12        a[i] = b[i] = i * 1.0;
13
14    chunk = CHUNKSIZE;
15    clock_t start = clock();
16    #pragma omp parallel for shared(a,b,c,chunk) private(i) schedule(dynamic/*static*/,chunk)
17    for (i=0; i < N; i++)
18    {
19        c[i] = a[i] + b[i];
20    }
21    clock_t end = clock();
22    double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
23
24    printf("for loop took %f seconds to execute \n", cpu_time_used);
25 }
```

The `*l6_2.c` tab is also visible and contains the same code. A terminal window is open in the foreground, showing the execution of the program:

```
ex1@AB1210SCOPE70: ~/Desktop/20BCE1143
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ gcc -o l62 -fopenmp l6_2.c
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ ./l62
for loop took 0.000410 seconds to execute
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ gcc -o l62 -fopenmp l6_2.c
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ ./l62
for loop took 0.001059 seconds to execute
ex1@AB1210SCOPE70:~/Desktop/20BCE1143$ 20BCE1143
```

(Code combined with next 3 questions)

Sample profile program with omp\_get\_wtime()



The screenshot shows a code editor with a file named `l6_3.c` open. The code is a C program that uses OpenMP for timing. It includes `<stdio.h>`, `<omp.h>`, and `<time.h>`. It defines `N` as 100000 and `CHUNKSIZE` as 100. The `main` function initializes an array `a` and then enters a parallel loop where each thread calculates `c[i] = a[i] + b[i]`. It uses `omp_get_wtime()` to measure the execution time of the loop. The terminal window shows the compilation and execution of the program, resulting in a time of 0.000614 seconds.

```
1 #include<stdio.h>
2 #include<omp.h>
3 #include<time.h>
4 #define N 100000
5 #define CHUNKSIZE 100
6 int main ()
7 {
8     int i, chunk;
9     float a[N], b[N], c[N];
10    /* Some initializations */
11    for (i=0; i < N; i++)
12        a[i] = b[i] = i * 1.0;
13
14    chunk = CHUNKSIZE;
15    //clock_t start = clock();
16    double start1 = omp_get_wtime();
17    #pragma omp parallel for shared(a,b,c,chunk) private(i) schedule(dynamic/*static*/,chunk)
18    for (i=0; i < N; i++)
19    {
20        c[i] = a[i] + b[i];
21    }
22    //clock_t end = clock();
23    //double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
24    double end1 = omp_get_wtime();
25
26    printf("for loop took %f seconds to execute \n", end1-start1);
27 }
28
```

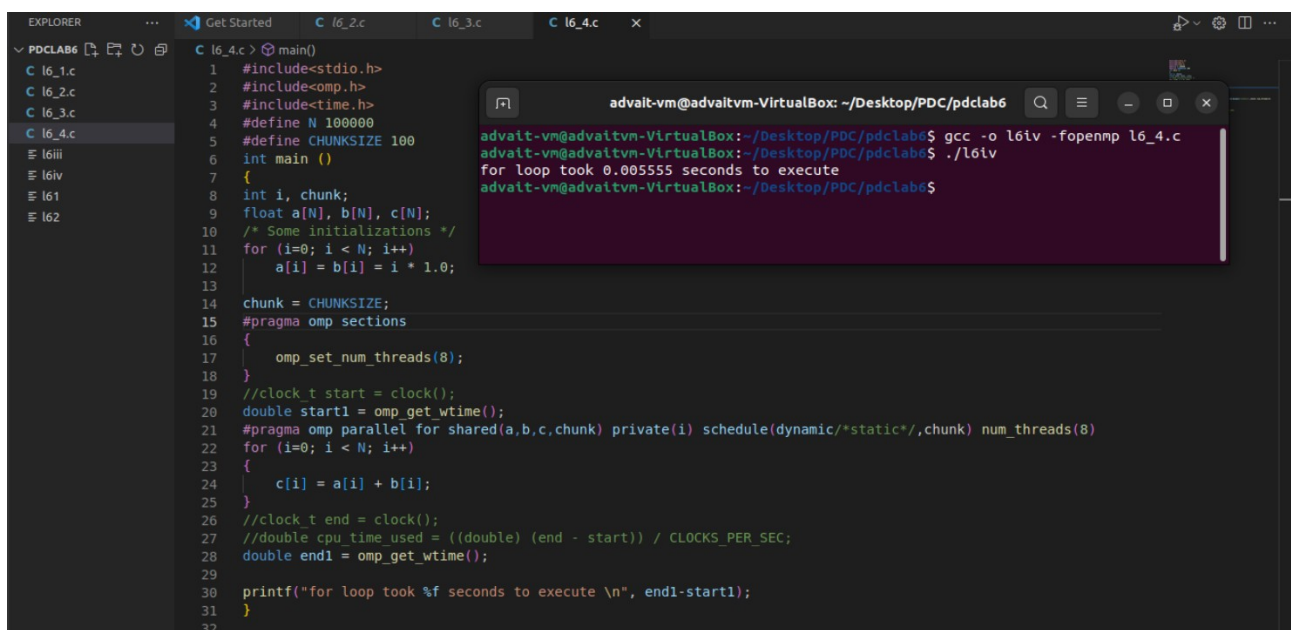
```
advait-vm@advaitvm-VirtualBox: ~/Desktop/PDC/pdclab6
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$ gcc -o l6iii -fopenmp l6_3.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$ ./l6iii
for loop took 0.000614 seconds to execute
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$
```

(Code combined with next 2 questions)

Sample program to set threads using directive clause

Sample program to set threads using omp\_set\_threads()

(Combined Together, as that is legal programming)



The screenshot shows a code editor with a file named `l6_4.c` open. The code is a C program that uses OpenMP for timing and thread control. It includes `<stdio.h>`, `<omp.h>`, and `<time.h>`. It defines `N` as 100000 and `CHUNKSIZE` as 100. The `main` function initializes an array `a` and then enters a parallel loop where each thread calculates `c[i] = a[i] + b[i]`. It uses `omp_set_num_threads(8)` to set the number of threads to 8. It uses `omp_get_wtime()` to measure the execution time of the loop. The terminal window shows the compilation and execution of the program, resulting in a time of 0.005555 seconds.

```
1 #include<stdio.h>
2 #include<omp.h>
3 #include<time.h>
4 #define N 100000
5 #define CHUNKSIZE 100
6 int main ()
7 {
8     int i, chunk;
9     float a[N], b[N], c[N];
10    /* Some initializations */
11    for (i=0; i < N; i++)
12        a[i] = b[i] = i * 1.0;
13
14    chunk = CHUNKSIZE;
15    #pragma omp sections
16    {
17        omp_set_num_threads(8);
18    }
19    //clock_t start = clock();
20    double start1 = omp_get_wtime();
21    #pragma omp parallel for shared(a,b,c,chunk) private(i) schedule(dynamic/*static*/,chunk) num_threads(8)
22    for (i=0; i < N; i++)
23    {
24        c[i] = a[i] + b[i];
25    }
26    //clock_t end = clock();
27    //double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
28    double end1 = omp_get_wtime();
29
30    printf("for loop took %f seconds to execute \n", end1-start1);
31 }
32
```

```
advait-vm@advaitvm-VirtualBox: ~/Desktop/PDC/pdclab6
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$ gcc -o l6iv -fopenmp l6_4.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$ ./l6iv
for loop took 0.005555 seconds to execute
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab6$
```

Code: Combined

```
#include<stdio.h>
#include<omp.h>
#include<time.h>
#define N 100000
#define CHUNKSIZE 100
int main ()
{
    int 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 sections
    {
        omp_set_num_threads(8);
        //clock_t start = clock();
        double start1 = omp_get_wtime();
        #pragma omp parallel for shared(a,b,c,chunk) private(i)
        schedule(dynamic/*static*/,chunk) num_threads(8)
        for (i=0; i < N; i++)
        {
            c[i] = a[i] + b[i];
        }
        //clock_t end = clock();
        //double cpu_time_used = ((double) (end - start)) /
        CLOCKS_PER_SEC;
        double end1 = omp_get_wtime();

        printf("for loop took %f seconds to execute \n", end1-start1);
    }
}
```