NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 4

02nd September 2020

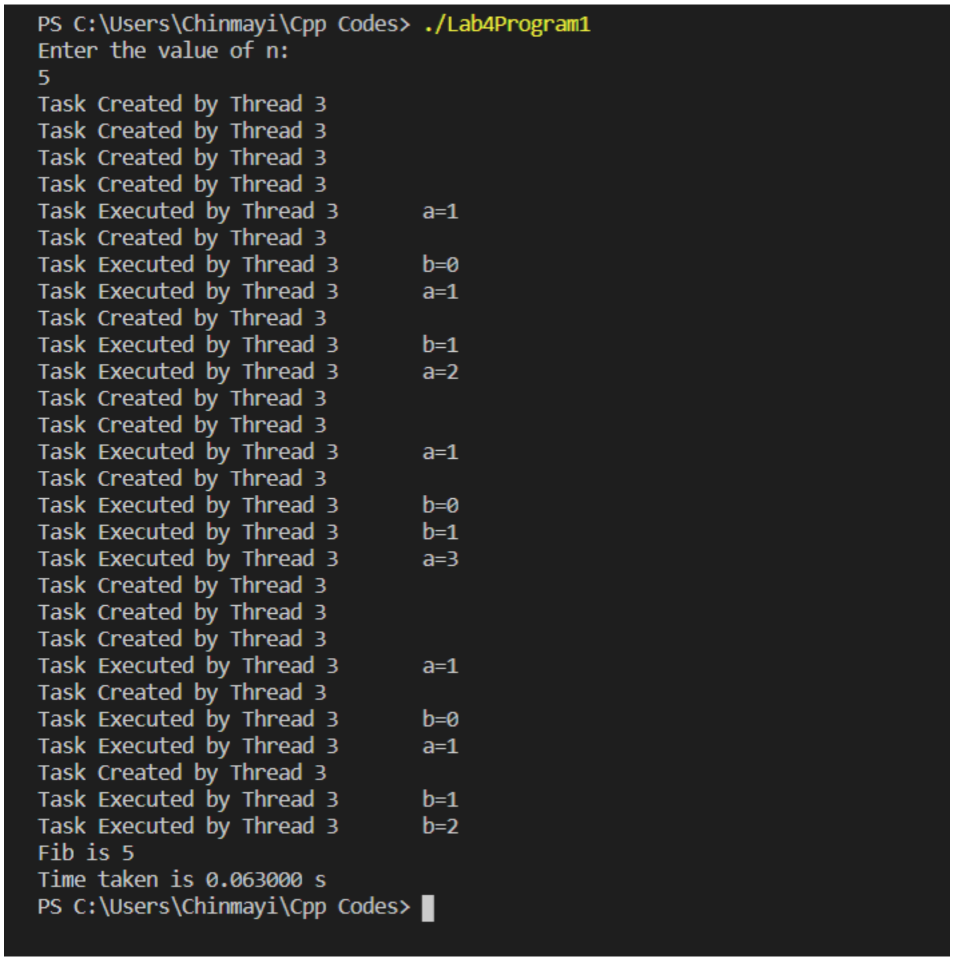
Faculty: Dr. Geetha V and Mrs. Tanmayee

Name: Chinmayi C. Ramakrishna

Roll No.: 181IT113

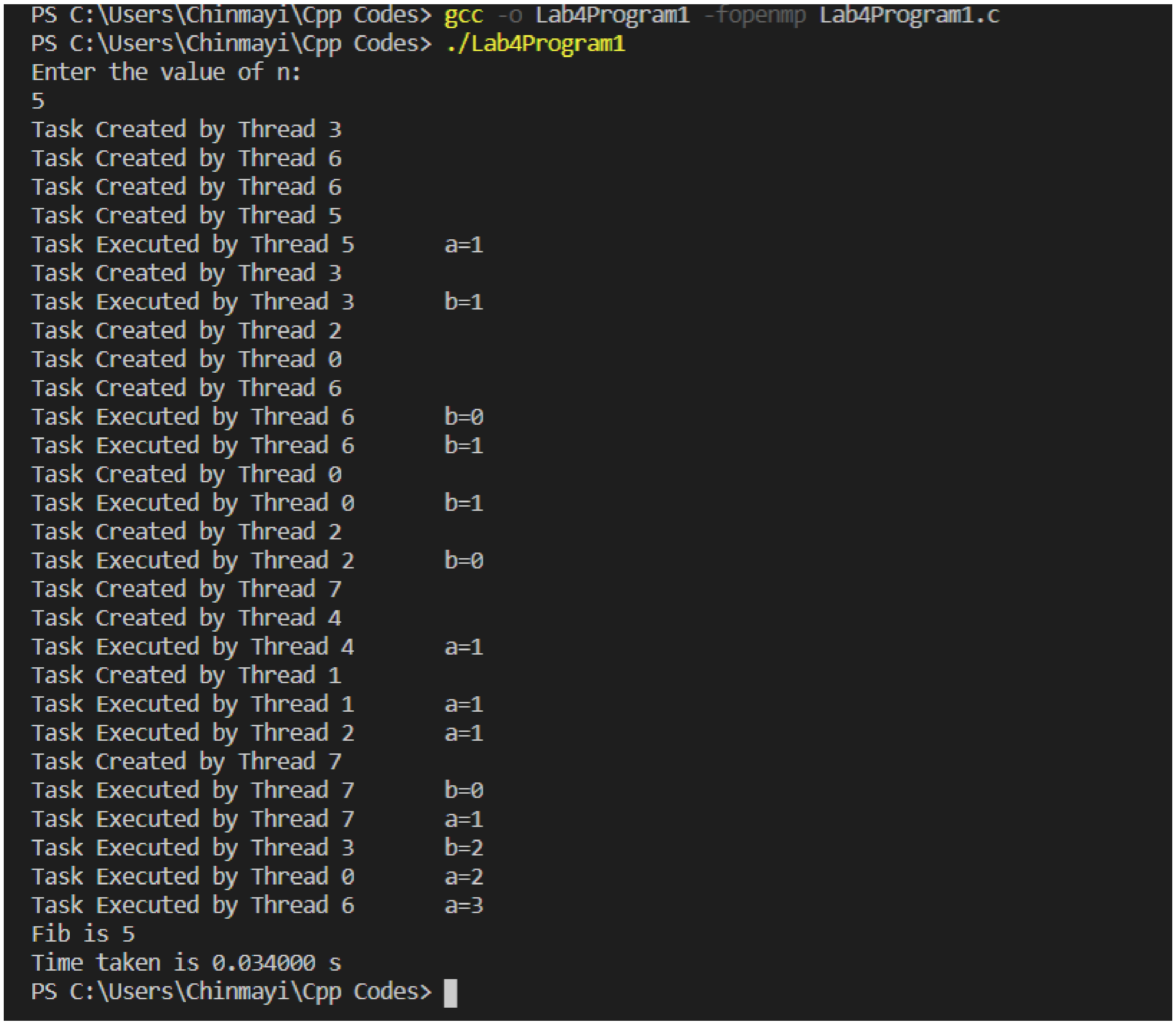
Program 1

Execute following code and observe the working of task directive.
Check the result by removing if () clause with task.



```
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program1
Enter the value of n:
5
Task Created by Thread 3
Task Created by Thread 3
Task Created by Thread 3
Task Created by Thread 3
Task Executed by Thread 3       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=0
Task Executed by Thread 3       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=1
Task Executed by Thread 3       a=2
Task Created by Thread 3
Task Created by Thread 3
Task Executed by Thread 3       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=0
Task Executed by Thread 3       b=1
Task Executed by Thread 3       a=3
Task Created by Thread 3
Task Created by Thread 3
Task Created by Thread 3
Task Executed by Thread 3       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=0
Task Executed by Thread 3       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=1
Task Executed by Thread 3       b=2
Fib is 5
Time taken is 0.063000 s
PS C:\Users\Chinmayi\Cpp Codes> █
```

Parallel execution with if () clause. Thread 3 generates an undeferred task and the task region isn’t resumed till the generated undeferred task completion.

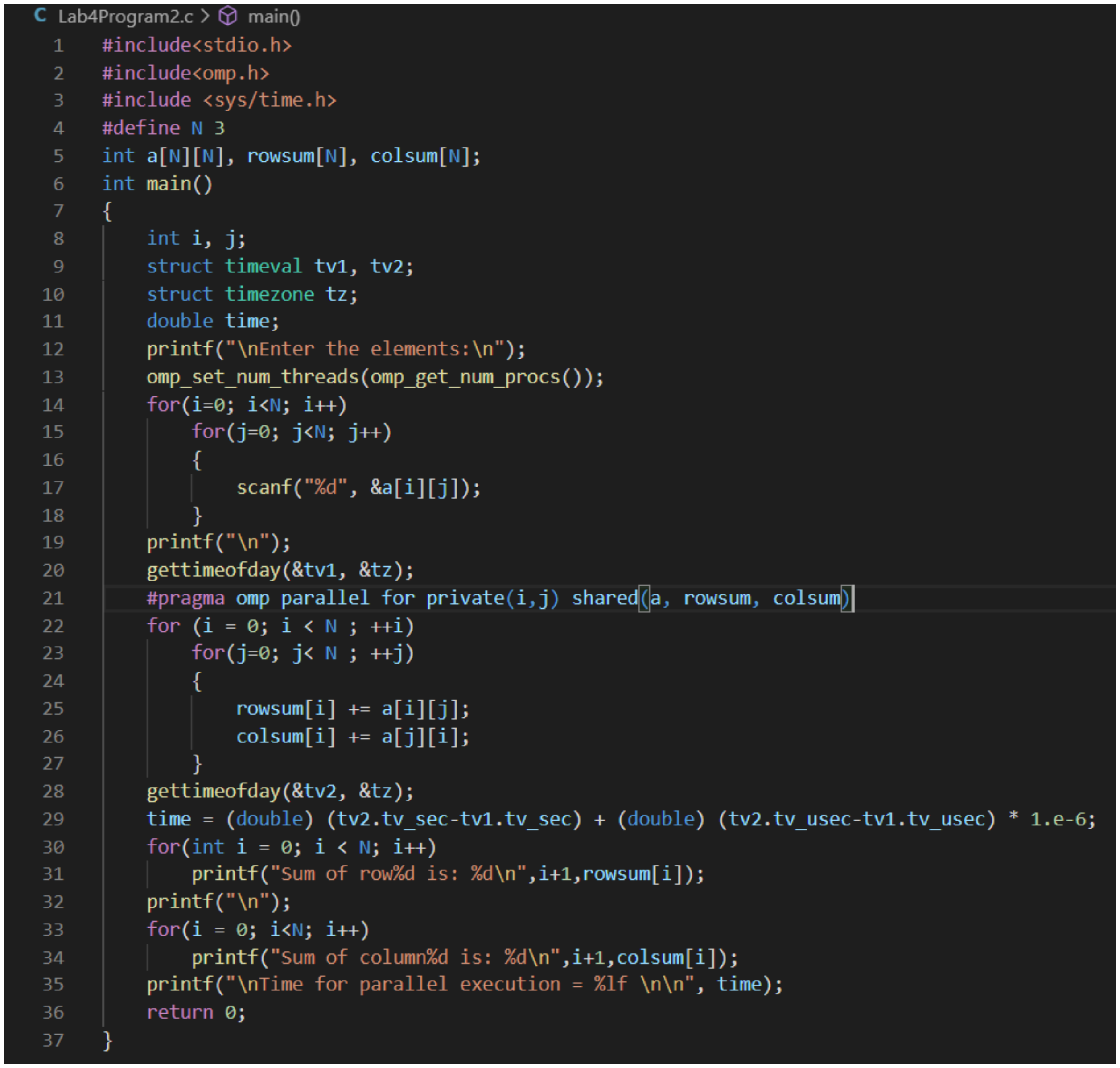


```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program1 -fopenmp Lab4Program1.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program1
Enter the value of n:
5
Task Created by Thread 3
Task Created by Thread 6
Task Created by Thread 6
Task Created by Thread 5
Task Executed by Thread 5       a=1
Task Created by Thread 3
Task Executed by Thread 3       b=1
Task Created by Thread 2
Task Created by Thread 0
Task Created by Thread 6
Task Executed by Thread 6       b=0
Task Executed by Thread 6       b=1
Task Created by Thread 0
Task Executed by Thread 0       b=1
Task Created by Thread 2
Task Executed by Thread 2       b=0
Task Created by Thread 7
Task Created by Thread 4
Task Executed by Thread 4       a=1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Executed by Thread 2       a=1
Task Created by Thread 7
Task Executed by Thread 7       b=0
Task Executed by Thread 7       a=1
Task Executed by Thread 3       b=2
Task Executed by Thread 0       a=2
Task Executed by Thread 6       a=3
Fib is 5
Time taken is 0.034000 s
PS C:\Users\Chinmayi\Cpp Codes>
```

Parallel execution without if () clause. Thread 3 may immediately execute the task, or defer its execution. In the latter case, any thread in the team may be assigned the task. Here thread 5 is assigned the task.

Program 2:

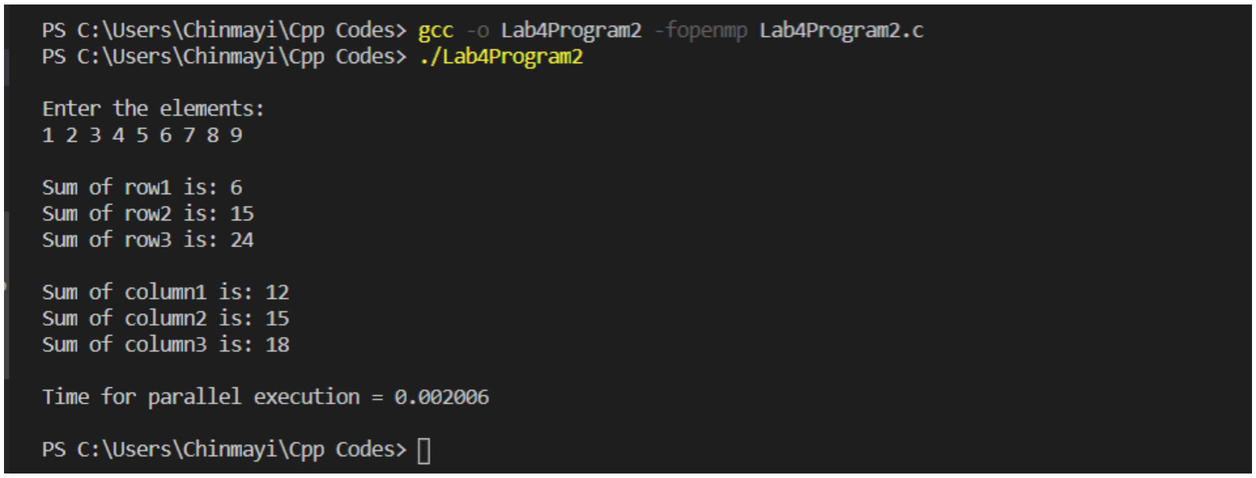
Write a C/C++ OpenMP program to find ROWSUM and COLUMNSUM of a matrix a[n][n]. Compare the time of parallel execution with sequential execution.



Lab4Program2.c > main()

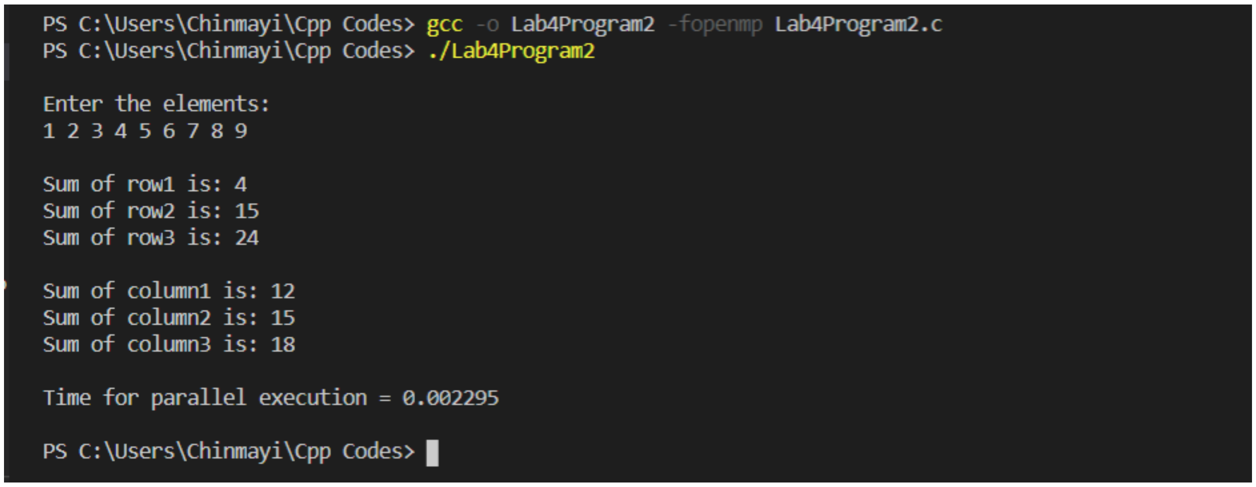
```c
#include<stdio.h>
#include<omp.h>
#include <sys/time.h>
#define N 3
int a[N][N], rowsum[N], colsum[N];
int main()
{
    int i, j;
    struct timeval tv1, tv2;
    struct timezone tz;
    double time;
    printf("\nEnter the elements:\n");
    omp_set_num_threads(omp_get_num_procs());
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
        {
            scanf("%d", &a[i][j]);
        }
    printf("\n");
    gettimeofday(&tv1, &tz);
    #pragma omp parallel for private(i,j) shared(a, rowsum, colsum)
    for (i = 0; i < N ; ++i)
        for(j=0; j< N ; ++j)
        {
            rowsum[i] += a[i][j];
            colsum[i] += a[j][i];
        }
    gettimeofday(&tv2, &tz);
    time = (double) (tv2.tv_sec-tv1.tv_sec) + (double) (tv2.tv_usec-tv1.tv_usec) * 1.e-6;
    for(int i = 0; i < N; i++)
        printf("Sum of row%d is: %d\n",i+1,rowsum[i]);
    printf("\n");
    for(i = 0; i<N; i++)
        printf("Sum of column%d is: %d\n",i+1,colsum[i]);
    printf("\nTime for parallel execution = %lf \n\n", time);
    return 0;
}
```

Output:



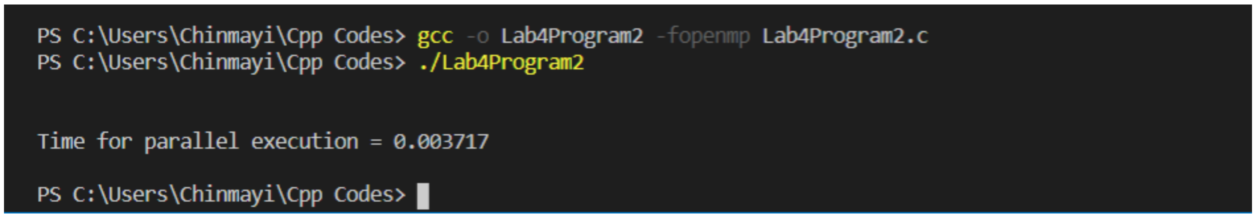
```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program2 -fopenmp Lab4Program2.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program2

Enter the elements:
1 2 3 4 5 6 7 8 9

Sum of row1 is: 6
Sum of row2 is: 15
Sum of row3 is: 24

Sum of column1 is: 12
Sum of column2 is: 15
Sum of column3 is: 18

Time for parallel execution = 0.002006

PS C:\Users\Chinmayi\Cpp Codes> 
```

Without collapse().



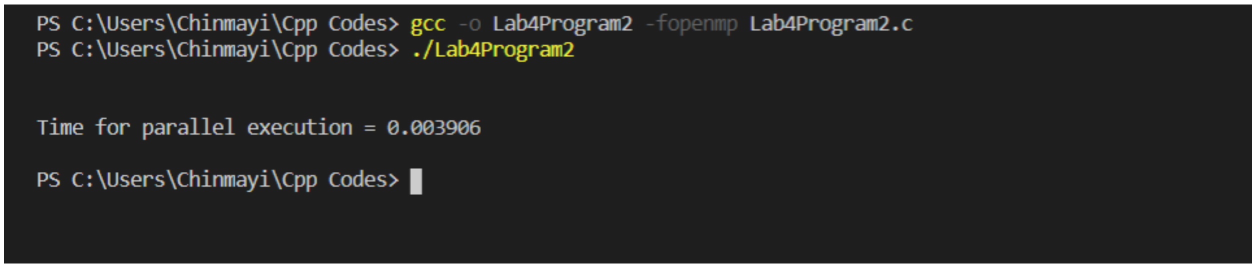
```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program2 -fopenmp Lab4Program2.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program2

Enter the elements:
1 2 3 4 5 6 7 8 9

Sum of row1 is: 4
Sum of row2 is: 15
Sum of row3 is: 24

Sum of column1 is: 12
Sum of column2 is: 15
Sum of column3 is: 18

Time for parallel execution = 0.002295

PS C:\Users\Chinmayi\Cpp Codes> 
```

Using collapse(2)



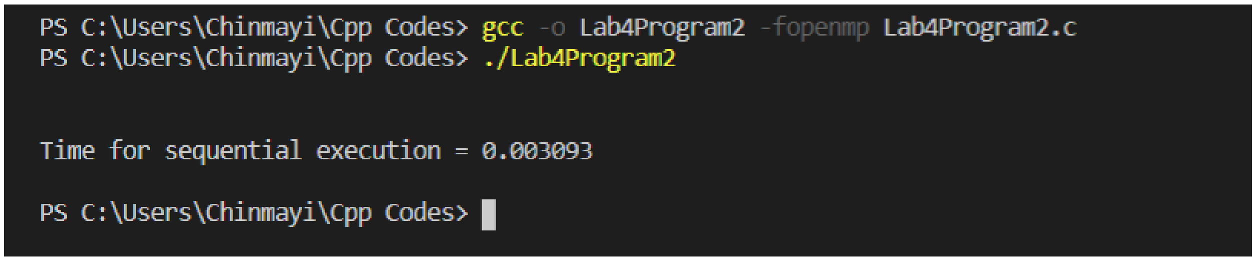
```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program2 -fopenmp Lab4Program2.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program2


Time for parallel execution = 0.003717

PS C:\Users\Chinmayi\Cpp Codes> 
```

Array size = 1000



```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program2 -fopenmp Lab4Program2.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program2


Time for parallel execution = 0.003906

PS C:\Users\Chinmayi\Cpp Codes>
```

Array size = 1000 and collapse (2)

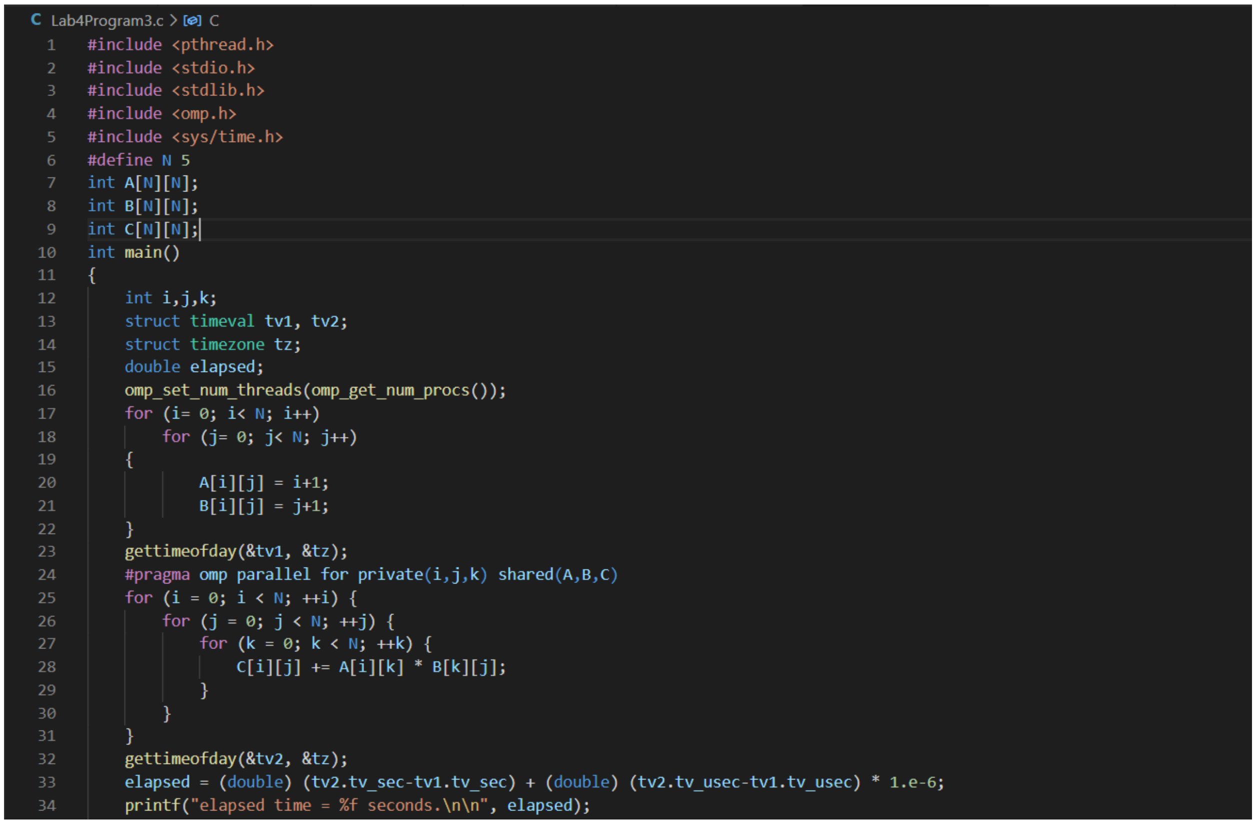


```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program2 -fopenmp Lab4Program2.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program2


Time for sequential execution = 0.003093

PS C:\Users\Chinmayi\Cpp Codes>
```

Array size = 1000 and sequential.

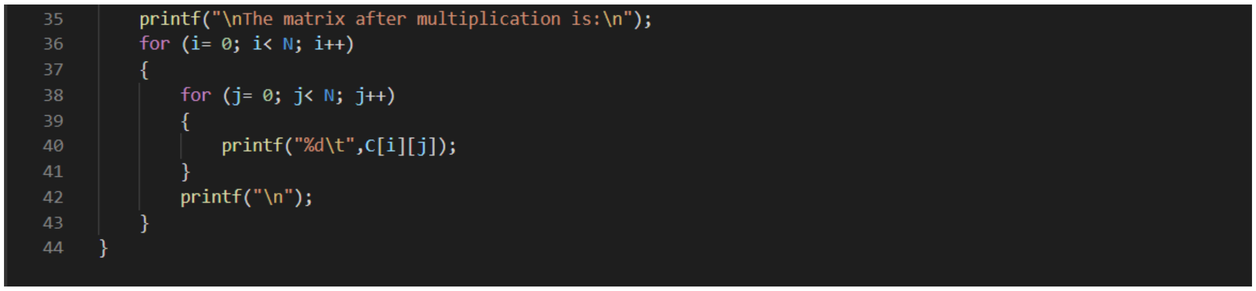
Program 3:

Write a C/C++ OpenMP program to perform matrix multiplication. Compare the time of parallel execution with sequential execution.



C Lab4Program3.c > [ø] C

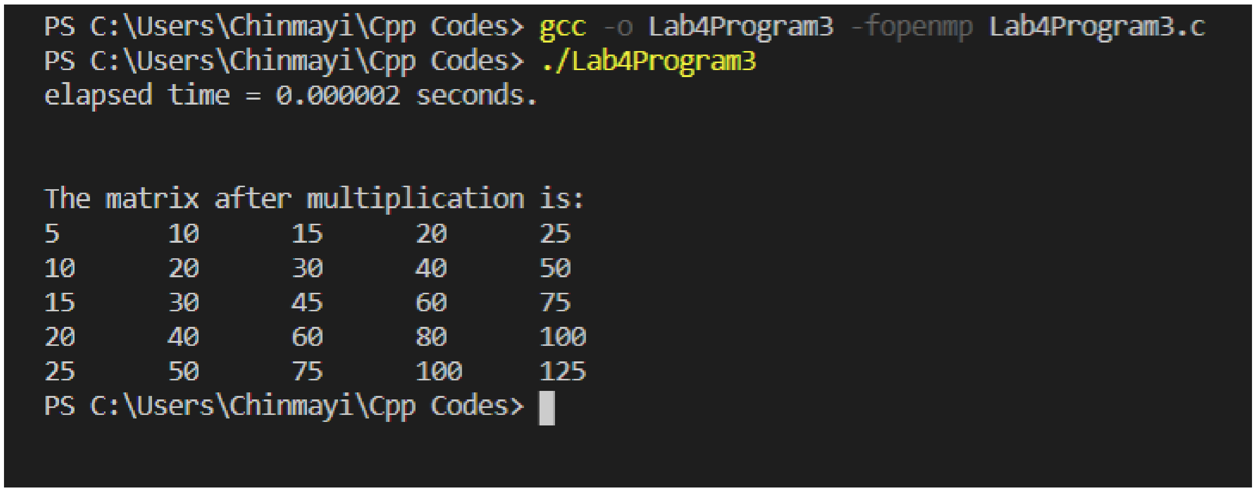
```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <sys/time.h>
#define N 5
int A[N][N];
int B[N][N];
int C[N][N];
int main()
{
    int i,j,k;
    struct timeval tv1, tv2;
    struct timezone tz;
    double elapsed;
    omp_set_num_threads(omp_get_num_procs());
    for (i= 0; i< N; i++)
        for (j= 0; j< N; j++)
    {
            A[i][j] = i+1;
            B[i][j] = j+1;
    }
    gettimeofday(&tv1, &tz);
    #pragma omp parallel for private(i,j,k) shared(A,B,C)
    for (i = 0; i < N; ++i) {
        for (j = 0; j < N; ++j) {
            for (k = 0; k < N; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    gettimeofday(&tv2, &tz);
    elapsed = (double) (tv2.tv_sec-tv1.tv_sec) + (double) (tv2.tv_usec-tv1.tv_usec) * 1.e-6;
    printf("elapsed time = %f seconds.\n\n", elapsed);
```



```c
    printf("\nThe matrix after multiplication is:\n");
    for (i= 0; i< N; i++)
    {
        for (j= 0; j< N; j++)
        {
            printf("%d\t",C[i][j]);
        }
        printf("\n");
    }
}
```

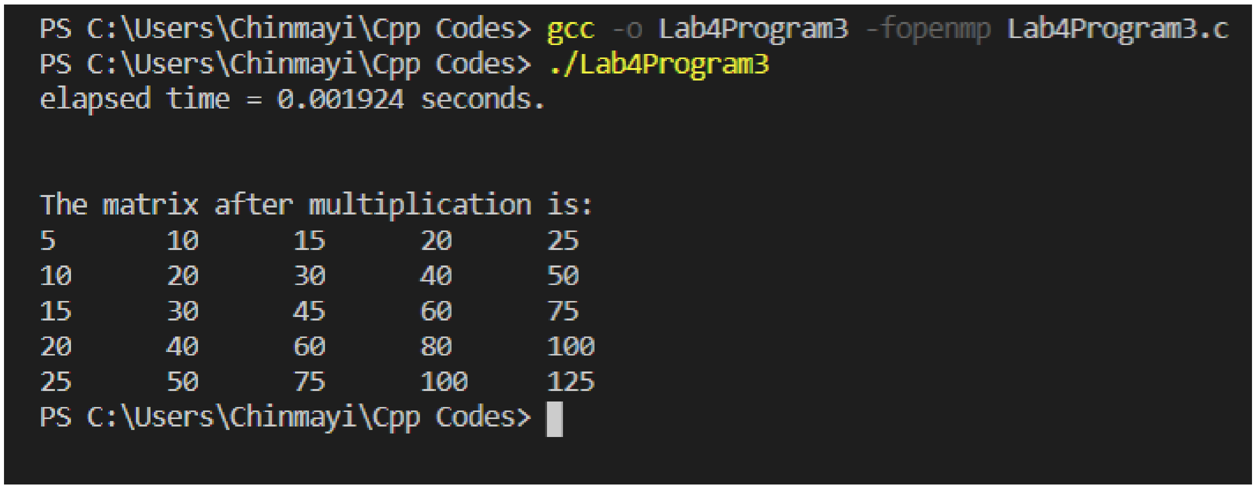
Code for matrix multiplication using parallel execution.

Output:



```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program3 -fopenmp Lab4Program3.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program3
elapsed time = 0.000002 seconds.


The matrix after multiplication is:
5       10      15      20      25
10      20      30      40      50
15      30      45      60      75
20      40      60      80      100
25      50      75      100     125
PS C:\Users\Chinmayi\Cpp Codes>
```

Matrix Multiplication using sequential execution.



```
PS C:\Users\Chinmayi\Cpp Codes> gcc -o Lab4Program3 -fopenmp Lab4Program3.c
PS C:\Users\Chinmayi\Cpp Codes> ./Lab4Program3
elapsed time = 0.001924 seconds.


The matrix after multiplication is:
5       10      15      20      25
10      20      30      40      50
15      30      45      60      75
20      40      60      80      100
25      50      75      100     125
PS C:\Users\Chinmayi\Cpp Codes>
```

Matrix Multiplication using parallel execution.

Sequential takes lesser time for smaller inputs.