Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Name : Kajal Jitendra Pawar

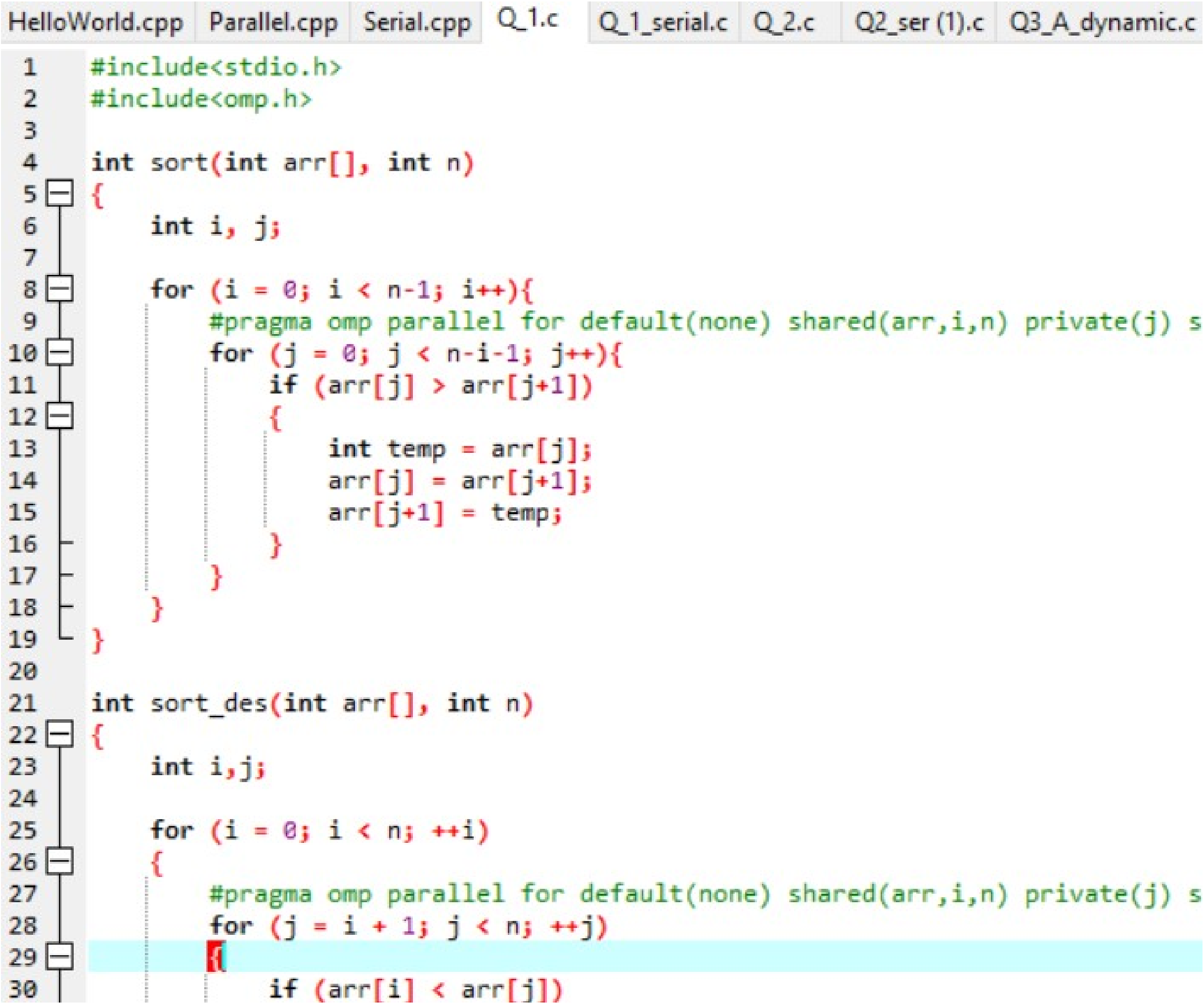
PRN : 2019BTECS00010

Batch : B1

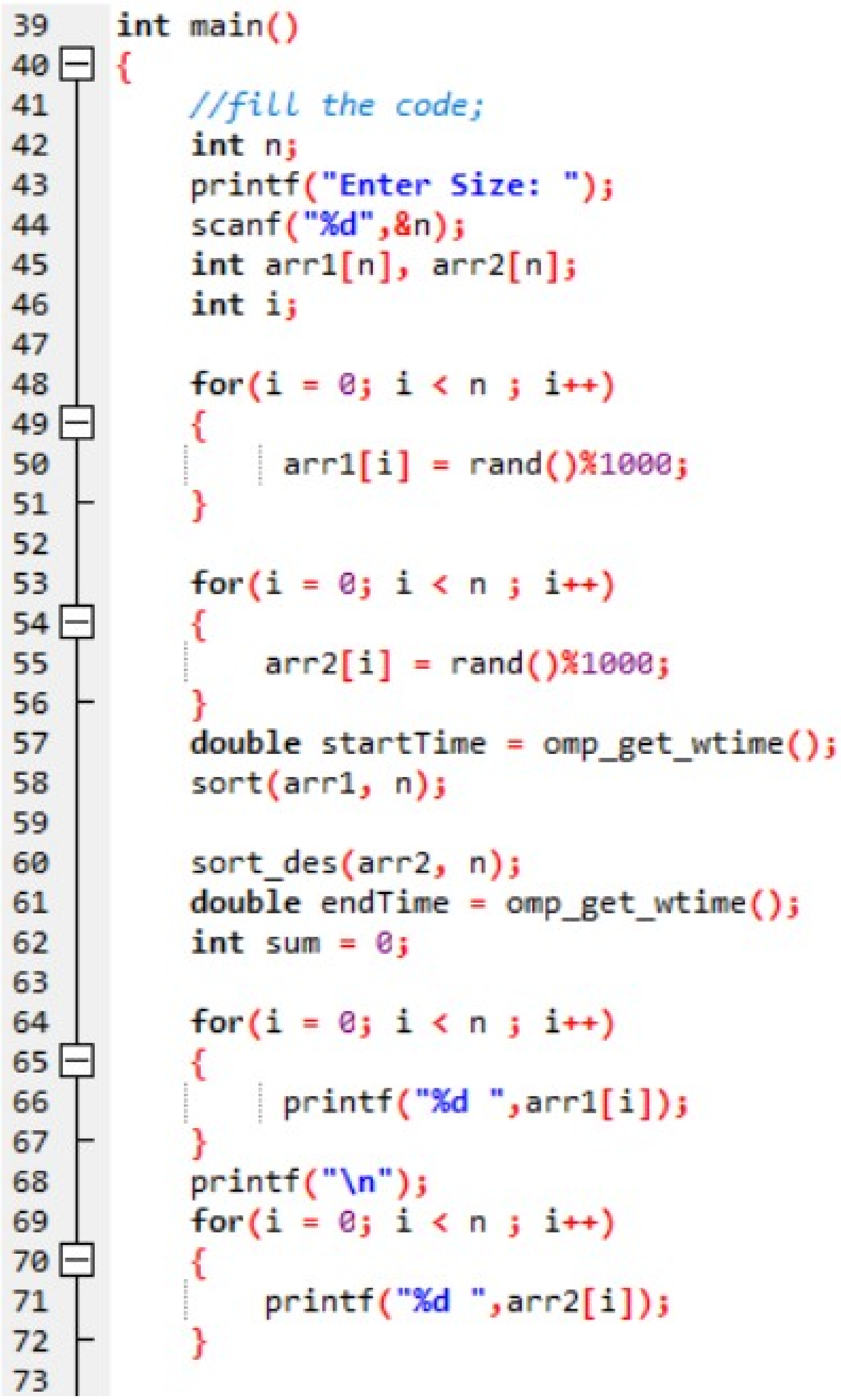
Practical No. 3

Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

Q1: Analyse and implement a Parallel code for below program using OpenMP.

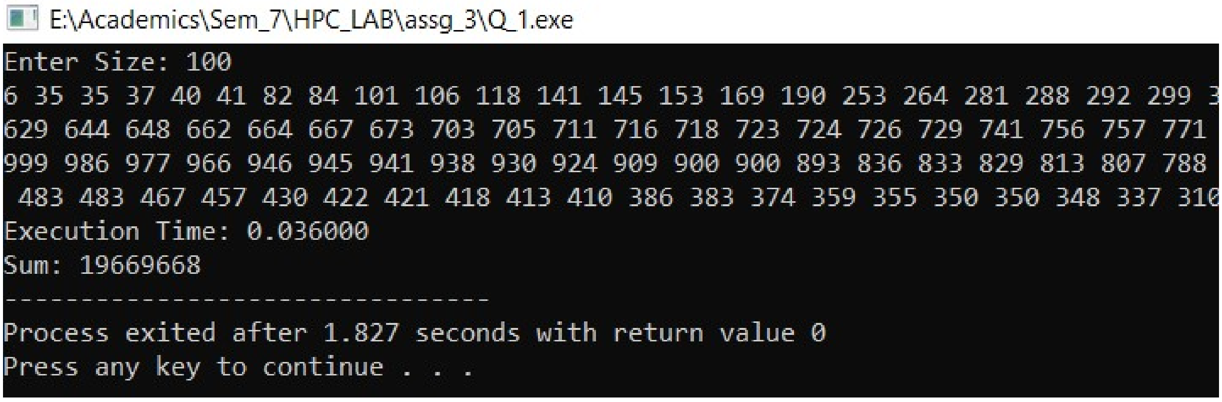


```c
#include<stdio.h>
#include<omp.h>

int sort(int arr[], int n)
{
    int i, j;

    for (i = 0; i < n-1; i++){
        #pragma omp parallel for default(none) shared(arr,i,n) private(j) s
        for (j = 0; j < n-i-1; j++){
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int sort_des(int arr[], int n)
{
    int i,j;

    for (i = 0; i < n; ++i)
    {
        #pragma omp parallel for default(none) shared(arr,i,n) private(j) s
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
```



```c
int main()
{
    //fill the code;
    int n;
    printf("Enter Size: ");
    scanf("%d",&n);
    int arr1[n], arr2[n];
    int i;

    for(i = 0; i < n ; i++)
    {
        arr1[i] = rand()%1000;
    }

    for(i = 0; i < n ; i++)
    {
        arr2[i] = rand()%1000;
    }
    double startTime = omp_get_wtime();
    sort(arr1, n);

    sort_des(arr2, n);
    double endTime = omp_get_wtime();
    int sum = 0;

    for(i = 0; i < n ; i++)
    {
        printf("%d ",arr1[i]);
    }
    printf("\n");
    for(i = 0; i < n ; i++)
    {
        printf("%d ",arr2[i]);
    }

```

Output :



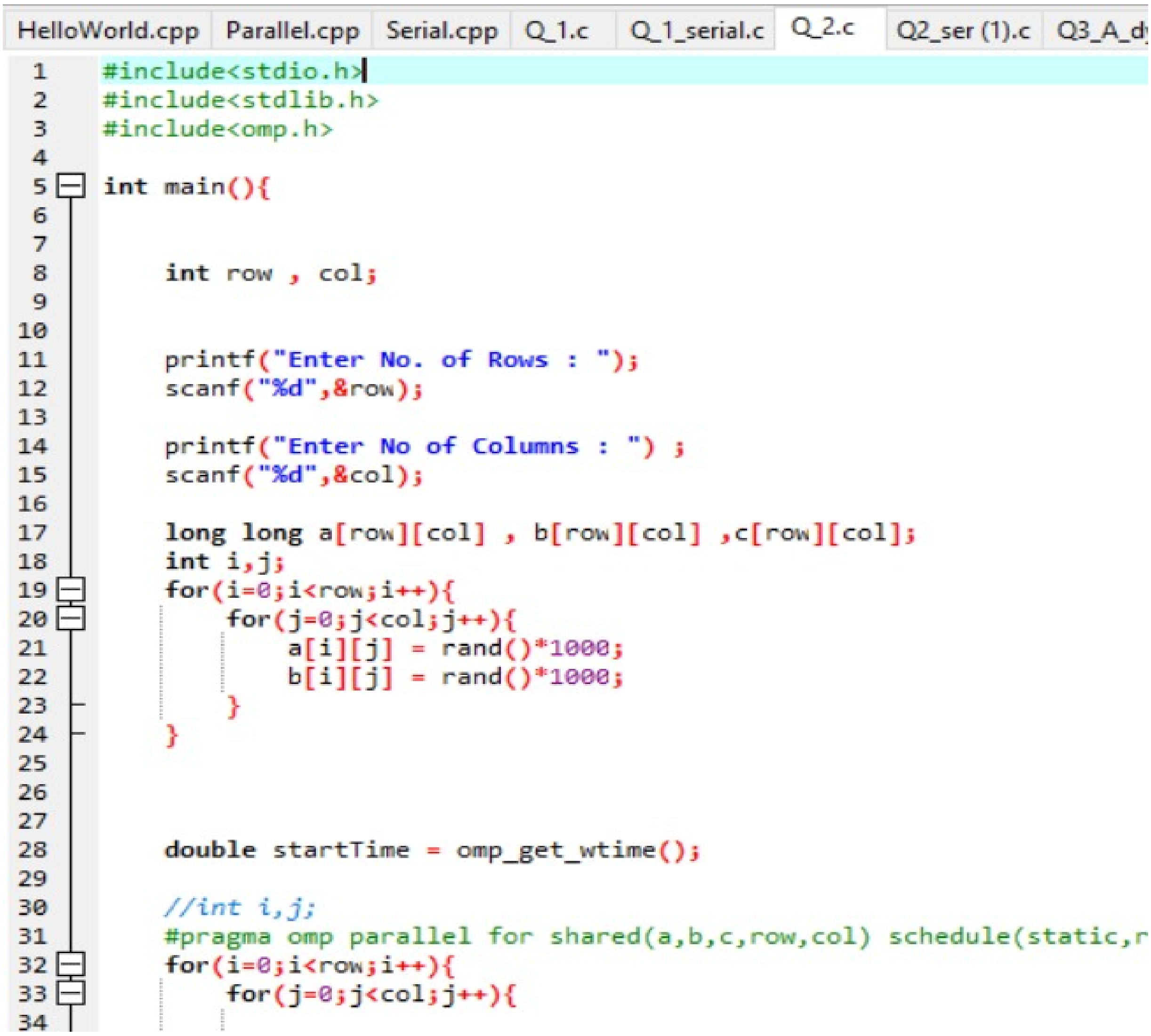
E:\Academics\Sem_7\HPC_LAB\assg_3\Q_1.exe

```
Enter Size: 100
6 35 35 37 40 41 82 84 101 106 118 141 145 153 169 190 253 264 281 288 292 299 3
629 644 648 662 664 667 673 703 705 711 716 718 723 724 726 729 741 756 757 771
999 986 977 966 946 945 941 938 930 924 909 900 900 893 836 833 829 813 807 788
 483 483 467 457 430 422 421 418 413 410 386 383 374 359 355 350 350 348 337 310
Execution Time: 0.036000
Sum: 19669668
--------------------------------
Process exited after 1.827 seconds with return value 0
Press any key to continue . . .
```

Information :
A schedule kind is passed to an OpenMP loop schedule clause: provides a hint for how iterations of the corresponding OpenMP loop should be assigned to threads in the team of the OpenMP region surrounding the loop.

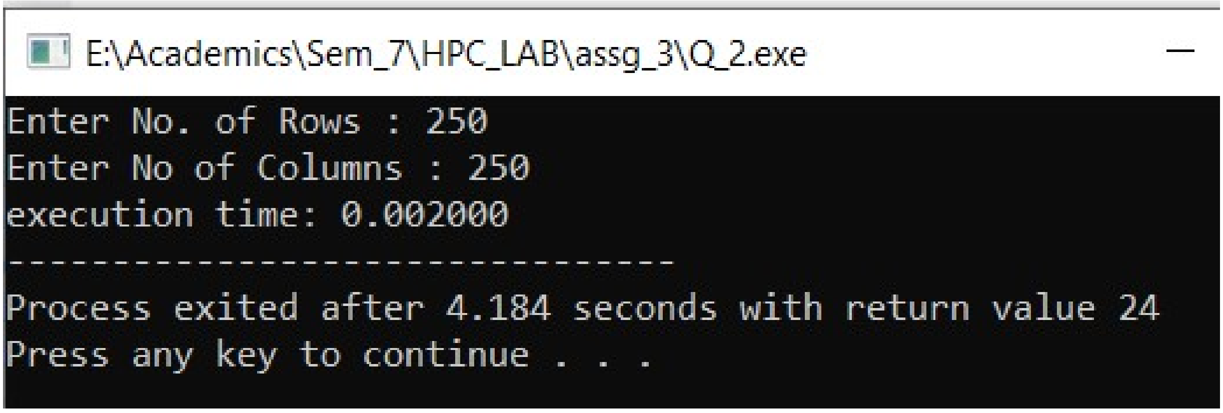
Q2: Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

i. **For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.**
ii. **Explain whether or not the scaling behaviour is as expected.**



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

int main(){


    int row , col;


    printf("Enter No. of Rows : ");
    scanf("%d",&row);

    printf("Enter No of Columns : ") ;
    scanf("%d",&col);

    long long a[row][col] , b[row][col] ,c[row][col];
    int i,j;
    for(i=0;i<row;i++){
        for(j=0;j<col;j++){
            a[i][j] = rand()*1000;
            b[i][j] = rand()*1000;
        }
    }



    double startTime = omp_get_wtime();

    //int i,j;
    #pragma omp parallel for shared(a,b,c,row,col) schedule(static,r
    for(i=0;i<row;i++){
        for(j=0;j<col;j++){

```

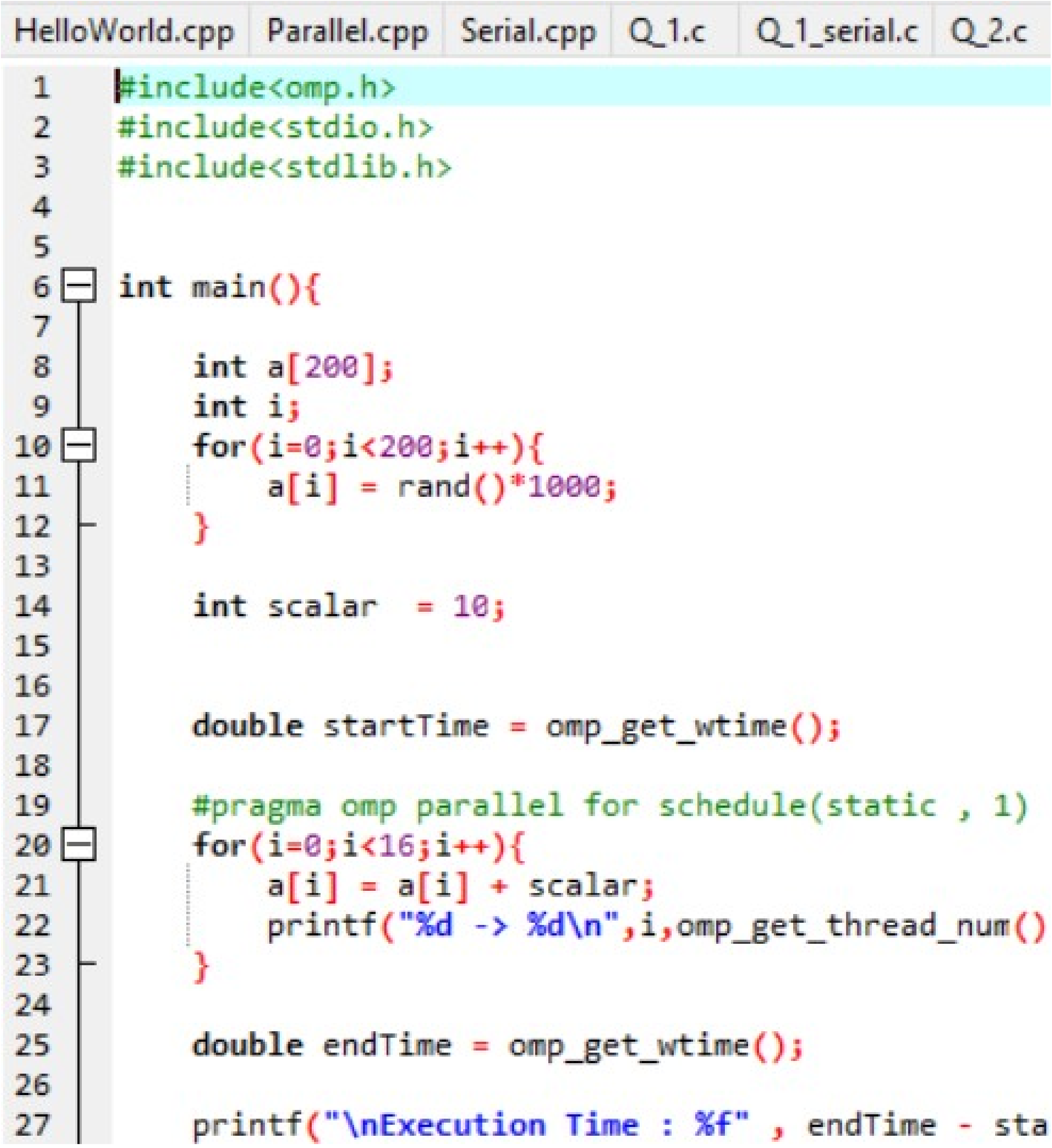
OUTPUT :



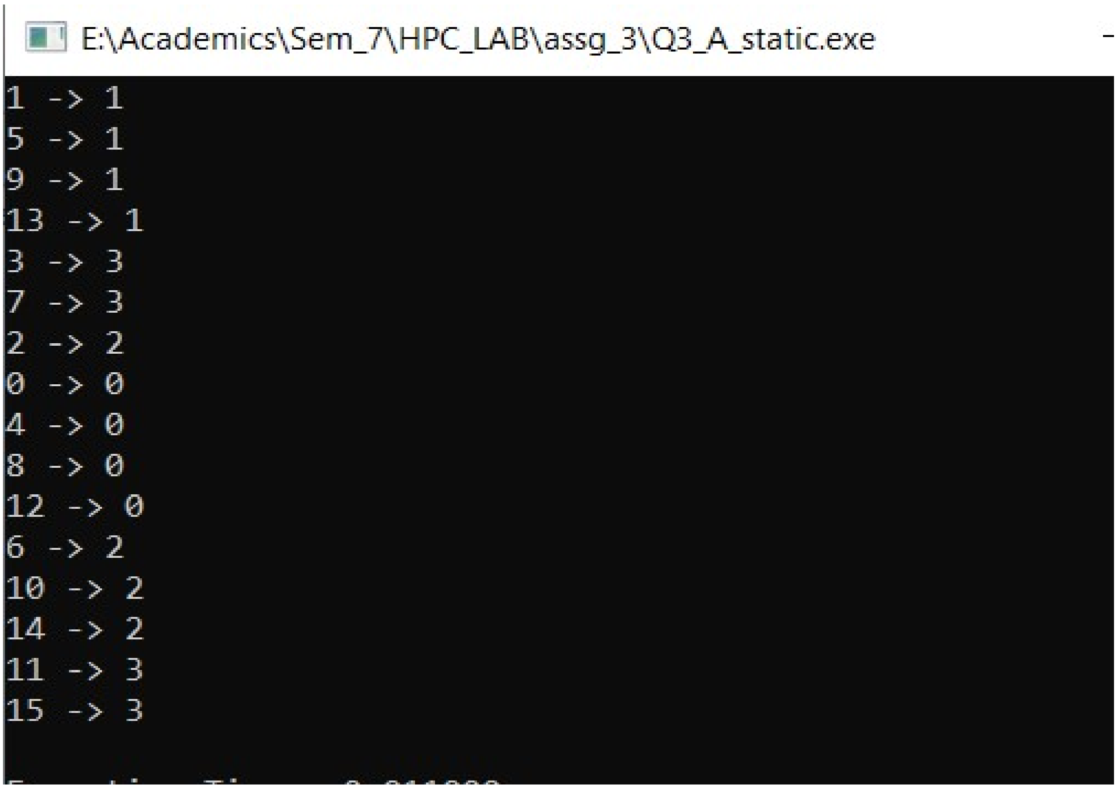
```
Enter No. of Rows : 250
Enter No of Columns : 250
execution time: 0.002000
--------------------------------
Process exited after 4.184 seconds with return value 24
Press any key to continue . . .
```

Q3. For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:

i. **Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.**

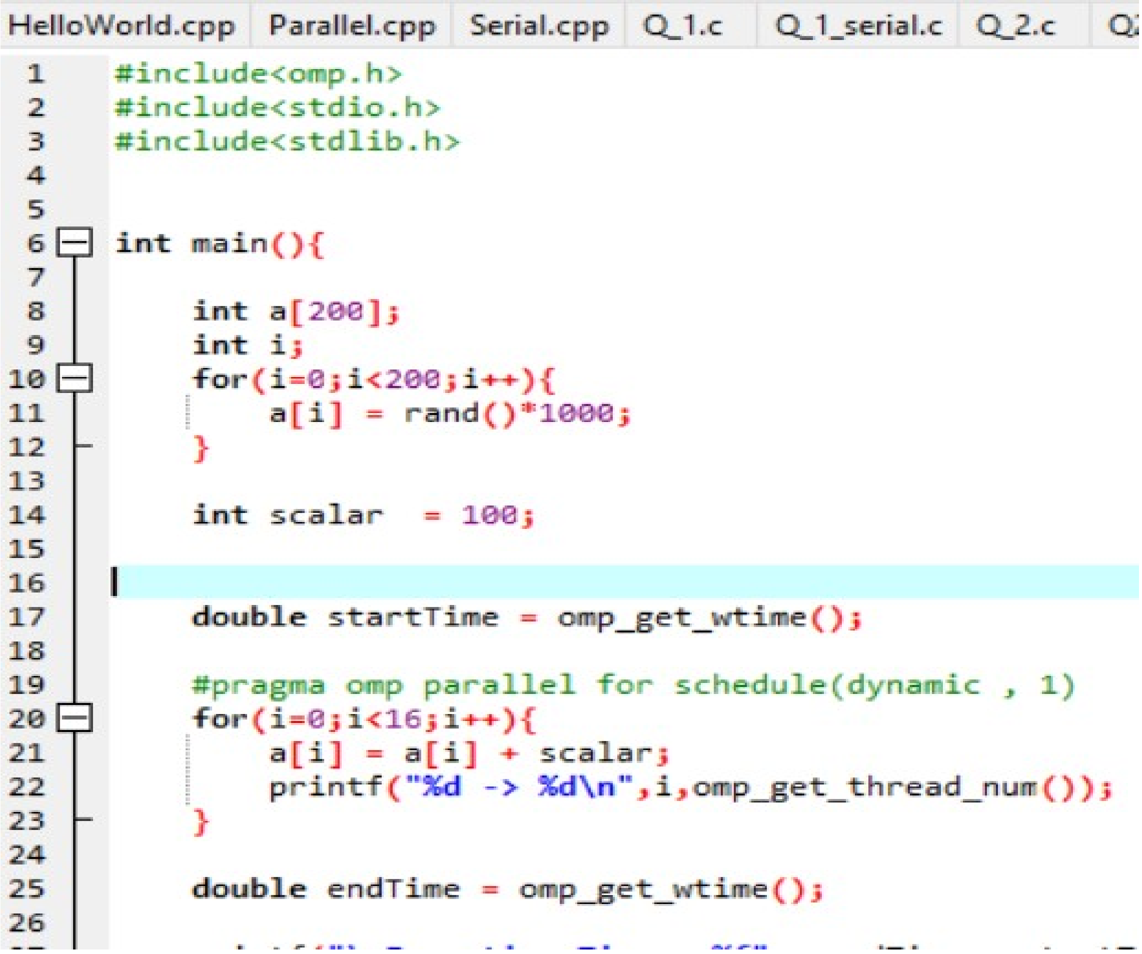


```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>


int main(){

    int a[200];
    int i;
    for(i=0;i<200;i++){
        a[i] = rand()*1000;
    }

    int scalar  = 10;


    double startTime = omp_get_wtime();

    #pragma omp parallel for schedule(static , 1)
    for(i=0;i<16;i++){
        a[i] = a[i] + scalar;
        printf("%d -> %d\n",i,omp_get_thread_num()
    }

    double endTime = omp_get_wtime();

    printf("\nExecution Time : %f" , endTime - sta
```

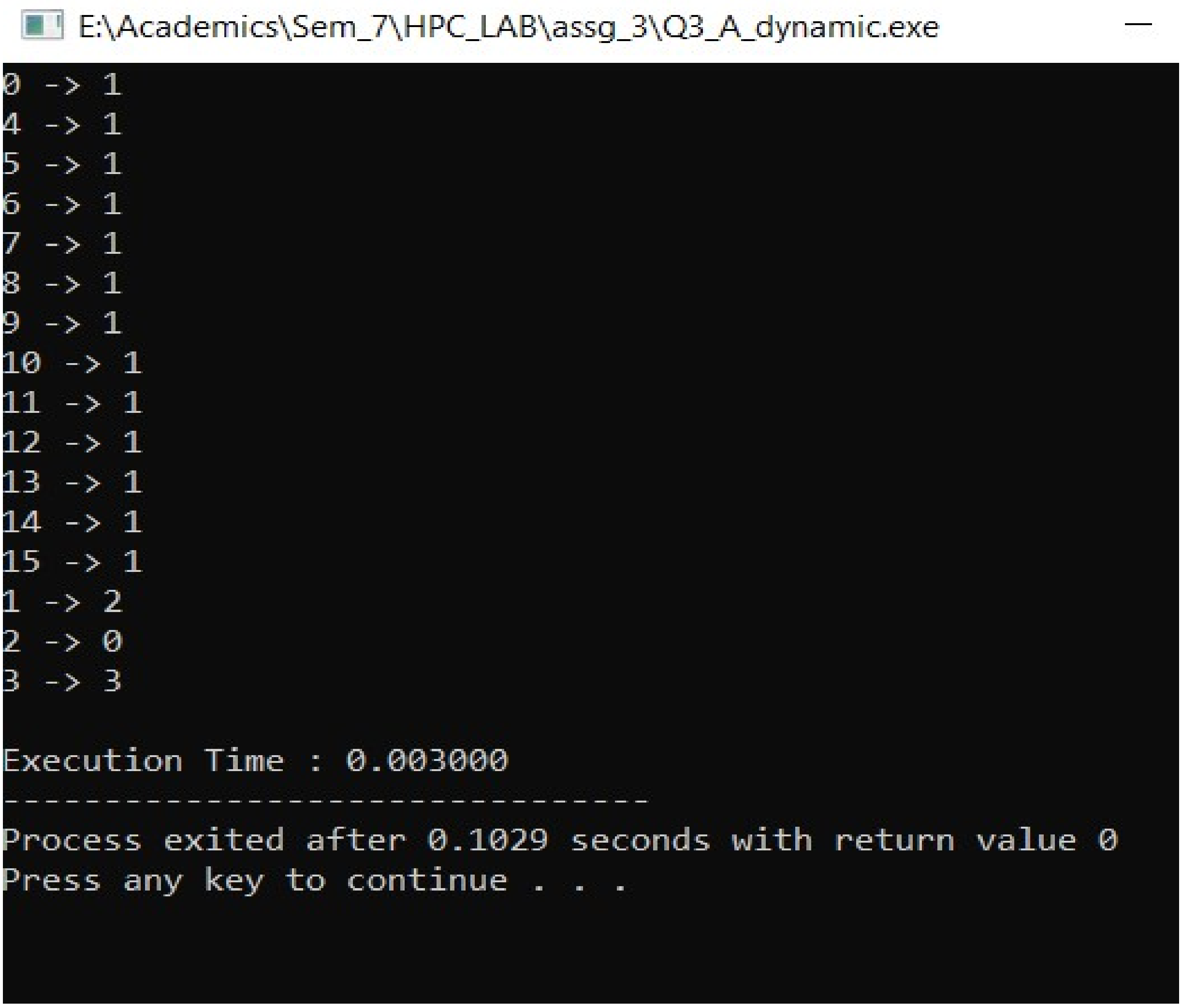


```
1 -> 1
5 -> 1
9 -> 1
13 -> 1
3 -> 3
7 -> 3
2 -> 2
0 -> 0
4 -> 0
8 -> 0
12 -> 0
6 -> 2
10 -> 2
14 -> 2
11 -> 3
15 -> 3
```

ii. **Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.**

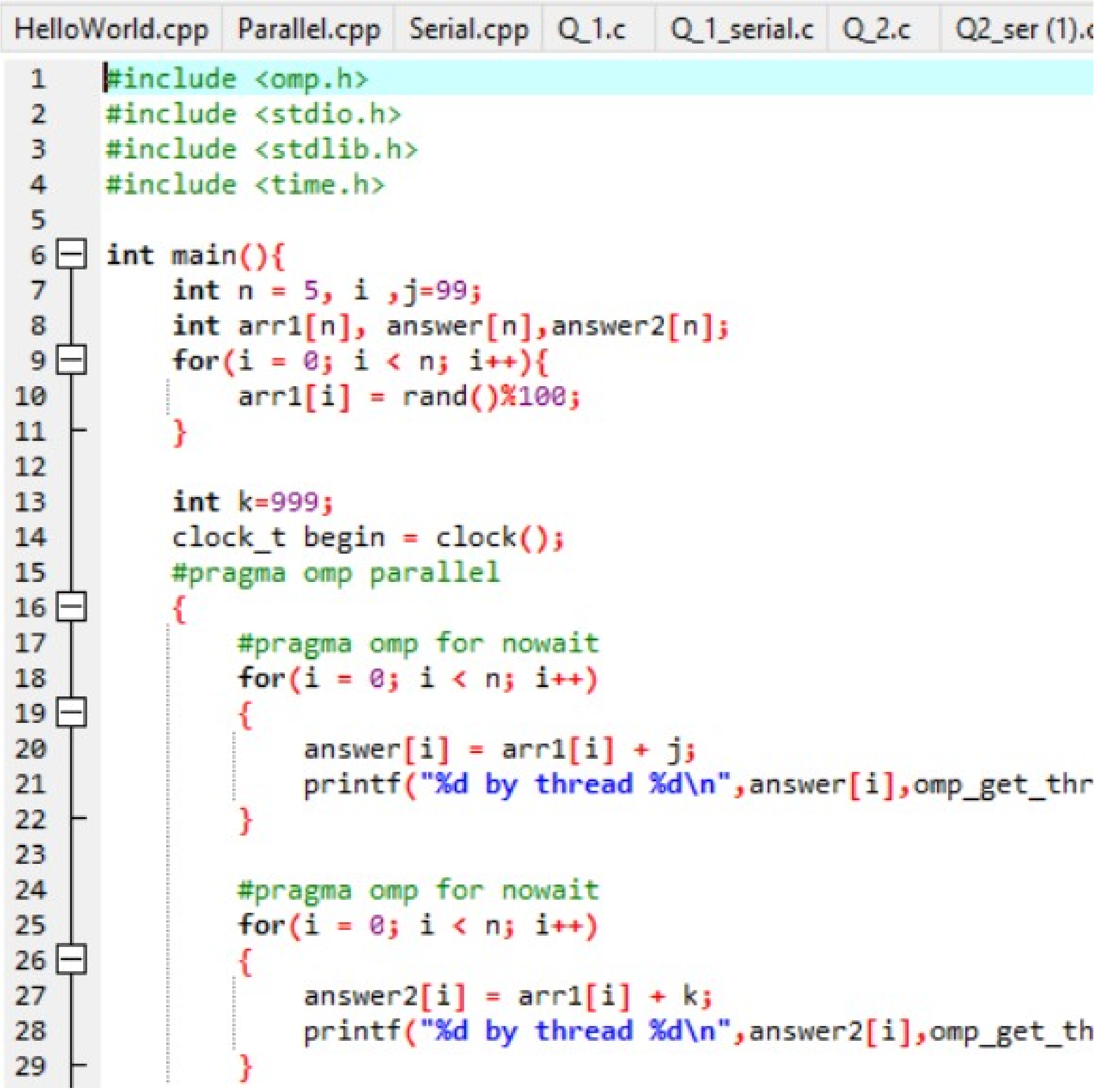


```c
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>


int main(){

	int a[200];
	int i;
	for(i=0;i<200;i++){
		a[i] = rand()*1000;
	}

	int scalar  = 100;


	double startTime = omp_get_wtime();

	#pragma omp parallel for schedule(dynamic , 1)
	for(i=0;i<16;i++){
		a[i] = a[i] + scalar;
		printf("%d -> %d\n",i,omp_get_thread_num());
	}

	double endTime = omp_get_wtime();

```

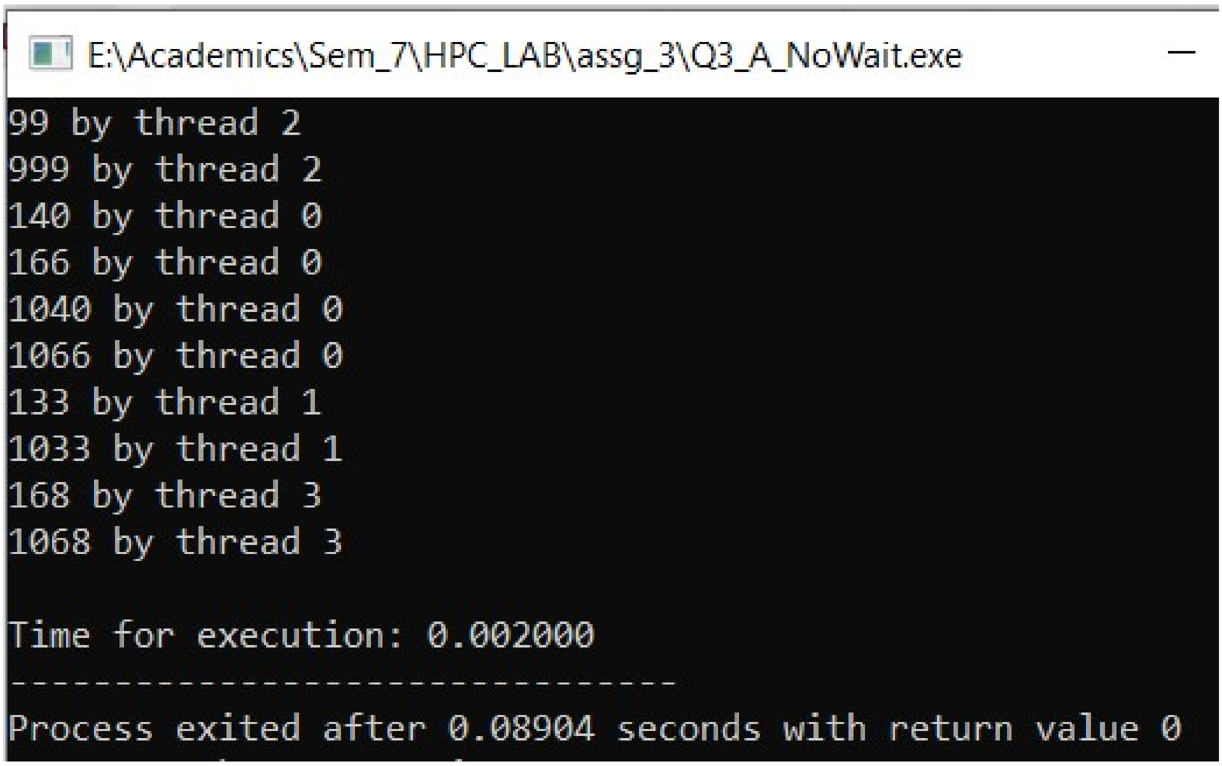


```
0 -> 1
4 -> 1
5 -> 1
6 -> 1
7 -> 1
8 -> 1
9 -> 1
10 -> 1
11 -> 1
12 -> 1
13 -> 1
14 -> 1
15 -> 1
1 -> 2
2 -> 0
3 -> 3

Execution Time : 0.003000
--------------------------------
Process exited after 0.1029 seconds with return value 0
Press any key to continue . . .
```

iii. **Demonstrate the use of nowait clause.**



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

int main(){
    int n = 5, i ,j=99;
    int arr1[n], answer[n],answer2[n];
    for(i = 0; i < n; i++){
        arr1[i] = rand()%100;
    }

    int k=999;
    clock_t begin = clock();
    #pragma omp parallel
    {
        #pragma omp for nowait
        for(i = 0; i < n; i++)
        {
            answer[i] = arr1[i] + j;
            printf("%d by thread %d\n",answer[i],omp_get_thr
        }

        #pragma omp for nowait
        for(i = 0; i < n; i++)
        {
            answer2[i] = arr1[i] + k;
            printf("%d by thread %d\n",answer2[i],omp_get_th
        }
```



```
99 by thread 2
999 by thread 2
140 by thread 0
166 by thread 0
1040 by thread 0
1066 by thread 0
133 by thread 1
1033 by thread 1
168 by thread 3
1068 by thread 3

Time for execution: 0.002000
--------------------------------
Process exited after 0.08904 seconds with return value 0
```

Static Schedules
By default, OpenMP statically assigns loop iterations to threads. When the parallel for block is entered, it assigns each thread the set of loop iterations it is to execute.

Dynamic Schedules:
The dynamic schedule is characterized by the property that no thread waits at the barrier for longer than it takes another thread to execute its final iteration.