Final Year B. Tech., Sem VII 2022-23

High Performance Computing Lab

PRN/ Roll No: 2020BTECS00206

Full Name: SAYALI YOGESH DESAI

Batch: B4

Assignment No. 3

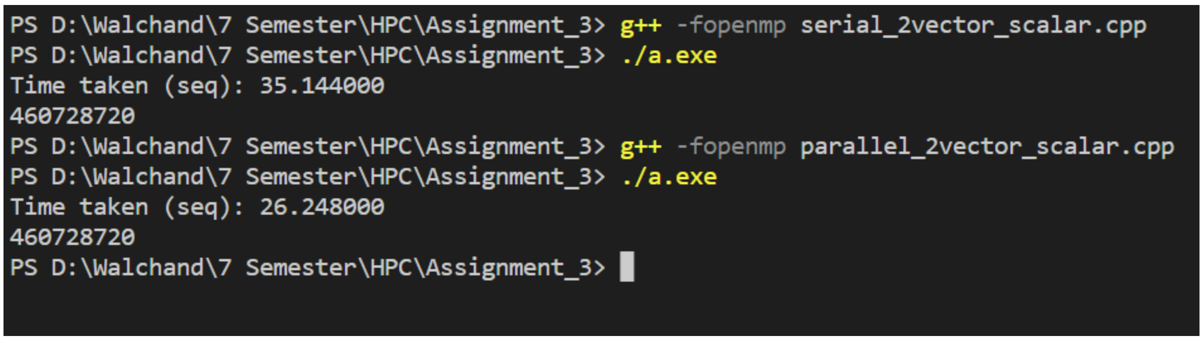
Que 1. Analyse and implement a parallel code for below program using openMP

***Sequential:**

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include <stdio.h>
#include <time.h>
#define n 100000
int sort(int arr[])
{
   int i, j;
   for (i = 0; i < n - 1; i++)
   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 i, j;
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
}
int main()
{
    int arr1[n], arr2[n];
    int i;
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr1[i]);
        arr1[i] = n - i;
    }
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr2[i]);
        arr2[i] = i;
    }
    clock_t t;
    t = clock();
    sort(arr1);
    sort_des(arr2);
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time_taken);
    int sum = 0;
    for (i = 0; i < n; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d\n", sum);
    return 0;
}
```

<u>*Parallel:</u>

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include <stdio.h>
#include <time.h>
#include<omp.h>
#define n 100000
int sort(int arr[])
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        int turn = i % 2;
        #pragma omp parallel for
        for (j = turn; j < n - 1; j+=2)
        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 i, j;
    for (i = 0; i < n; ++i)
    {
        int turn = i % 2;
        #pragma omp parallel for
        for (j = turn; j < n - 1; j += 2)
        {
            // printf("Thread ID: ",omp_get_thread_num());
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main()
{
    int arr1[n], arr2[n];
    int i;
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr1[i]);
        arr1[i] = n - i;
    }
    for (i = 0; i < n; i++)
    {
        //scanf("%d", &arr2[i]);
        arr2[i] = i;
    }
    clock_t t;
    t = clock();
    sort(arr1);
    sort_des(arr2);
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("Time taken (seq): %f\n", time_taken);
    int sum = 0;
    for (i = 0; i < n; i++)
    {
        // printf("%d %d\n", arr1[i],arr2[i]);
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("%d\n", sum);
    return 0;
}
```



```
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp serial_2vector_scalar.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Time taken (seq): 35.144000
460728720
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp parallel_2vector_scalar.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Time taken (seq): 26.248000
460728720
PS D:\Walchand\7 Semester\HPC\Assignment_3>
```

Que 2. 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.**

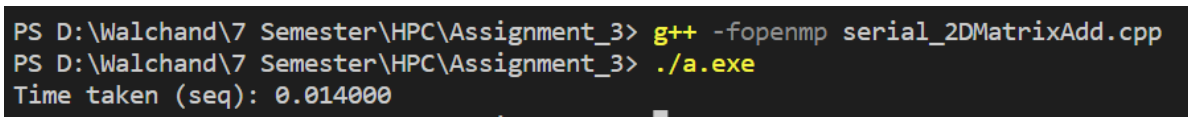
ii. **Explain whether or not the scaling behaviour is as expected.**

<u>*Sequential-</u>

- **Code:**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 2000
void add(int **a, int **b, int **c)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
void input(int **a, int num)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            a[i][j] = num;
        }
    }
}
void display(int **a)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
int main()
{
    int **a = (int **)malloc(sizeof(int *) * N);
    int **b = (int **)malloc(sizeof(int *) * N);
    int **c = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++)
    {
        a[i] = (int *)malloc(sizeof(int) * N);
        b[i] = (int *)malloc(sizeof(int) * N);
        c[i] = (int *)malloc(sizeof(int) * N);
    }
    input(a, 1);
    input(b, 1);
    double start = omp_get_wtime();
    add(a, b, c);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
```

- **Output:**



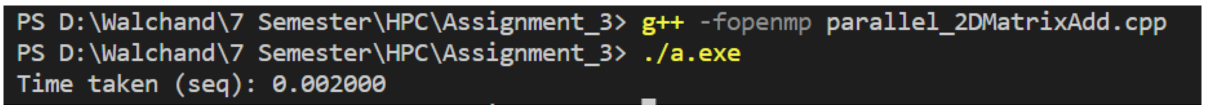
```
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp serial_2DMatrixAdd.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Time taken (seq): 0.014000
```

<u>*Parallel:</u>

- **Code-**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 1000
void add(int **a, int **b, int **c) {
#pragma omp parallel for
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
void input(int **a, int num) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            a[i][j] = num;
        }
    }
}
void displayMatrix(int **a) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}


int main() {
    int **a = (int **)malloc(sizeof(int *) * N);
    int **b = (int **)malloc(sizeof(int *) * N);
    int **c = (int **)malloc(sizeof(int *) * N);
    for (int i = 0; i < N; i++) {
        a[i] = (int *)malloc(sizeof(int) * N);
        b[i] = (int *)malloc(sizeof(int) * N);
        c[i] = (int *)malloc(sizeof(int) * N);
    }
    input(a, 1);
    input(b, 1);
    double start = omp_get_wtime();
    add(a, b, c);
    double end = omp_get_wtime();
    // display(c);
    printf("Time taken (seq): %f\n", end - start);
}
```

- **Output:**



```
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp parallel_2DMatrixAdd.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Time taken (seq): 0.002000
```

Que 3. 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. Analyse the speedup.**
- **ii.** **Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyse the speedup.**
- **iii.** **Demonstrate the use of nowait clause**

<u>STATIC Schedule:</u>

```
#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 200

int main()

{

    int **a=(int **) malloc(sizeof(int) * N);

    int **c=(int **) malloc(sizeof(int) * N);

    int b = 10;

    omp_set_num_threads(6);

    for(int i=0; i<N; i++)

    {

        a[i] = 0;

    }

    double itime, ftime, exec_time;

    itime = omp_get_wtime();

    #pragma omp parallel for schedule(static, 8)

    for(int i=0; i<N; i++)

    {

        c[i] = a[i] + b;
    }
    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\n\nTime taken is %f\n", exec_time);
}
```

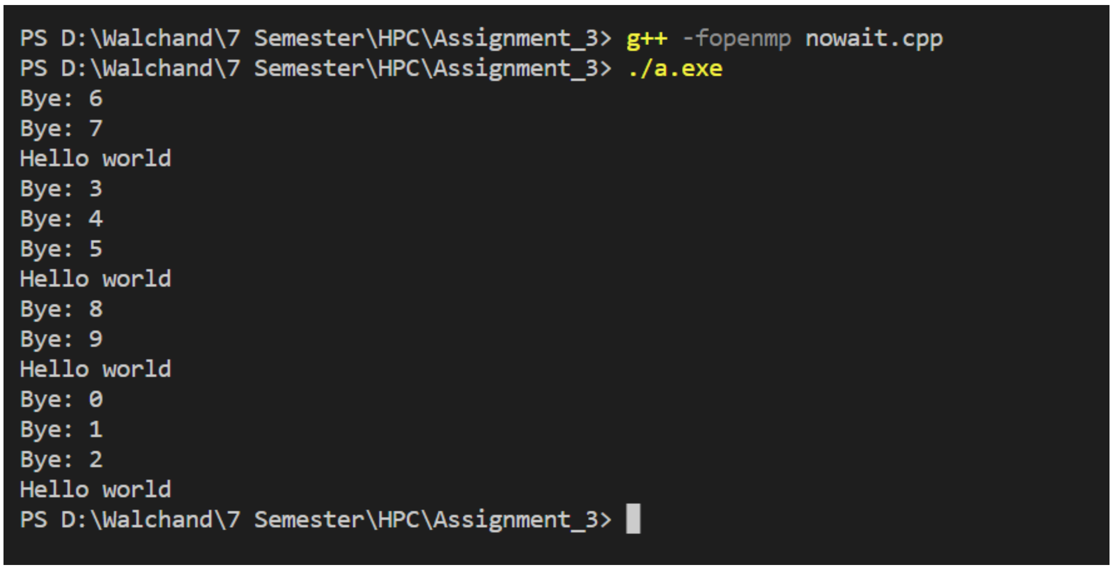
DYNAMIC Schedule:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 200
int main()
{
    int **a= (int **)malloc(sizeof(int) * N);
    int **c= (int **)malloc(sizeof(int) * N);
    int b = 10;
    omp_set_num_threads(6);
    for(int i=0; i<N; i++)
    {
        a[i] = 0;
    }
    double itime, ftime, exec_time;
    itime = omp_get_wtime();
    #pragma omp parallel for schedule(dynamic, 2)
    for(int i=0; i<N; i++)
    {
        c[i] = a[i] + b;
    }
    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\n\nTime taken is %f\n", exec_time);
}
```

<u>NOWAIT Clause:</u>

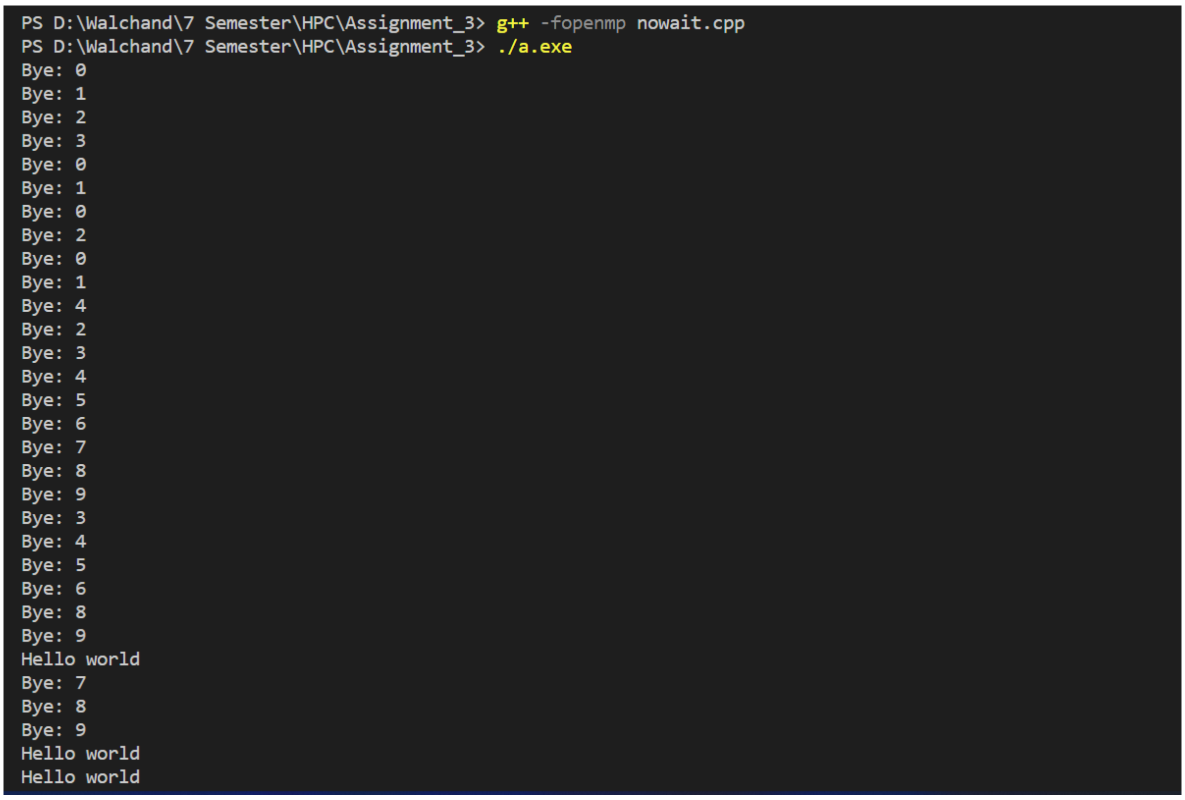
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 10
void hello_world()
{
    printf("Hello world\n");
}
void bye(int i)
{
    printf("Bye: %d\n", i);
}
int main()
{
    int *a = (int*)malloc(sizeof(int) * N);
    for(int i=0; i<N; i++)
    {
        a[i] = 1;
    }
    #pragma omp parallel

    {

    #pragma omp for nowait

    for(int i=0; i<N; i++)

    {

       bye(i);

    }

    hello_world();

    }

}
```



```
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp nowait.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Bye: 6
Bye: 7
Hello world
Bye: 3
Bye: 4
Bye: 5
Hello world
Bye: 8
Bye: 9
Hello world
Bye: 0
Bye: 1
Bye: 2
Hello world
PS D:\Walchand\7 Semester\HPC\Assignment_3>
```

Without NOWAIT Clause:



```
PS D:\Walchand\7 Semester\HPC\Assignment_3> g++ -fopenmp nowait.cpp
PS D:\Walchand\7 Semester\HPC\Assignment_3> ./a.exe
Bye: 0
Bye: 1
Bye: 2
Bye: 3
Bye: 0
Bye: 1
Bye: 0
Bye: 2
Bye: 0
Bye: 1
Bye: 4
Bye: 2
Bye: 3
Bye: 4
Bye: 5
Bye: 6
Bye: 7
Bye: 8
Bye: 9
Bye: 3
Bye: 4
Bye: 5
Bye: 6
Bye: 8
Bye: 9
Hello world
Bye: 7
Bye: 8
Bye: 9
Hello world
Hello world
```