

Practical No 5

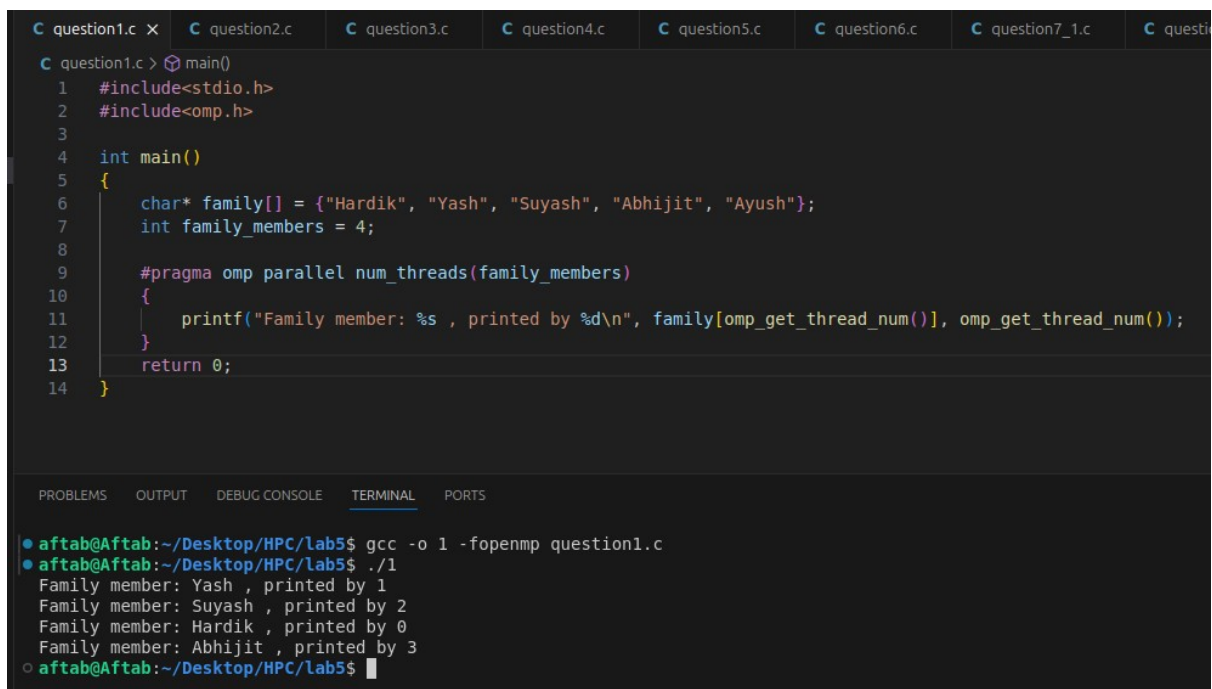
PRN: 22520005

Name: Aftab Imtiyaj Bhadgaonkar

Batch: B6

Course: High Performance Computing Lab

Q1. Write an OpenMP program such that, it should print the name of your family members, such that the names should come from different threads/cores. Also print the respective job id.

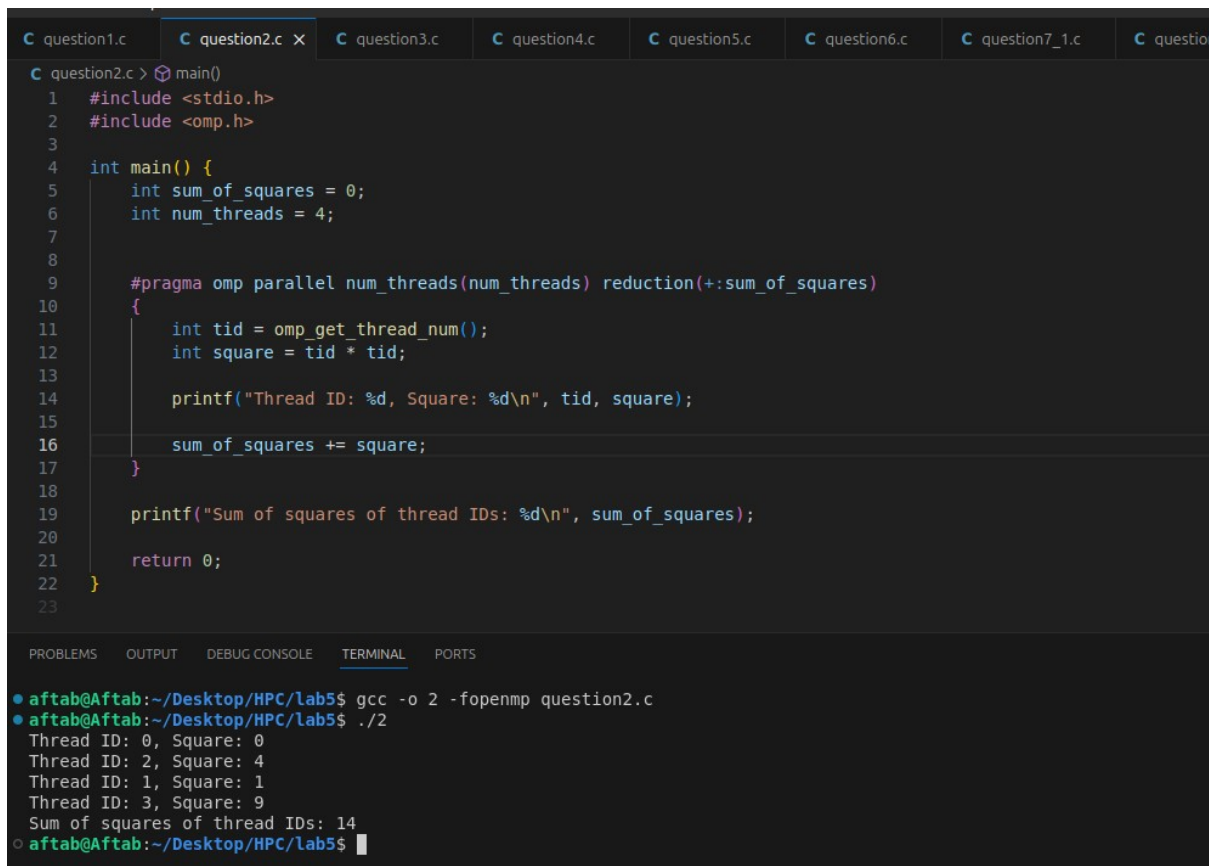


```
C question1.c x C question2.c C question3.c C question4.c C question5.c C question6.c C question7_1.c C question8.c
C question1.c > main()
1  #include<stdio.h>
2  #include<omp.h>
3
4  int main()
5  {
6      char* family[] = {"Hardik", "Yash", "Suyash", "Abhijit", "Ayush"};
7      int family_members = 4;
8
9      #pragma omp parallel num_threads(family_members)
10     {
11         printf("Family member: %s , printed by %d\n", family[omp_get_thread_num()], omp_get_thread_num());
12     }
13     return 0;
14 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 1 -fopenmp question1.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./1
Family member: Yash , printed by 1
Family member: Suyash , printed by 2
Family member: Hardik , printed by 0
Family member: Abhijit , printed by 3
○ aftab@Aftab:~/Desktop/HPC/lab5$
```

Q2. Write an OpenMP program such that, it should print the sum of square of the thread id's. Also make sure that, each thread should print the square value of their thread id.



The screenshot displays a code editor with a dark theme. At the top, there are tabs for various files: 'question1.c', 'question2.c' (active), 'question3.c', 'question4.c', 'question5.c', 'question6.c', 'question7_1.c', and 'question8.c'. The active tab shows the following C code:

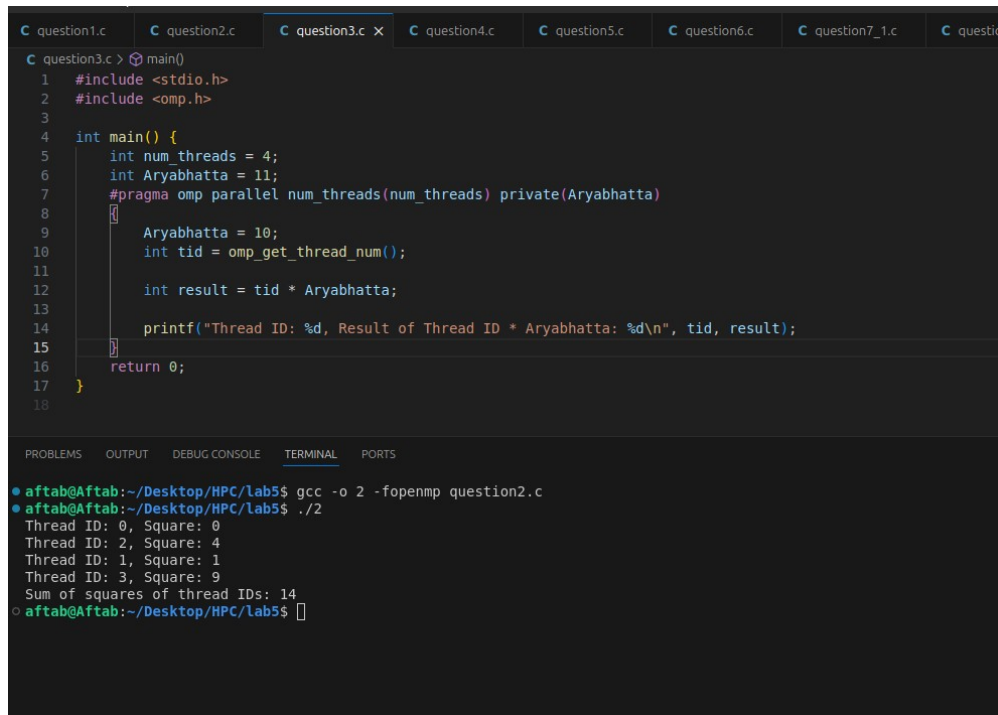
```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main() {
5      int sum_of_squares = 0;
6      int num_threads = 4;
7
8
9      #pragma omp parallel num_threads(num_threads) reduction(+:sum_of_squares)
10     {
11         int tid = omp_get_thread_num();
12         int square = tid * tid;
13
14         printf("Thread ID: %d, Square: %d\n", tid, square);
15
16         sum_of_squares += square;
17     }
18
19     printf("Sum of squares of thread IDs: %d\n", sum_of_squares);
20
21     return 0;
22 }
23
```

Below the code editor, there is a terminal window with the following output:

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 2 -fopenmp question2.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./2
Thread ID: 0, Square: 0
Thread ID: 2, Square: 4
Thread ID: 1, Square: 1
Thread ID: 3, Square: 9
Sum of squares of thread IDs: 14
○ aftab@Aftab:~/Desktop/HPC/lab5$
```

Q3. Consider a variable called “Aryabhata” declared as 10 (i.e int Aryabhata=10). Write an OpenMP program which should print the result of multiplication of thread id and value of the above variable.

Note*: The variable “Aryabhata” should be declared as private



The screenshot displays a code editor with a C program named 'question3.c' and a terminal window showing its execution. The program uses OpenMP to create 4 parallel threads. Each thread calculates the product of its thread ID and a private variable 'Aryabhata' (initialized to 10). The results are printed for each thread, and the sum of the squares of the thread IDs is calculated and printed.

```
C question3.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main() {
5      int num_threads = 4;
6      int Aryabhata = 11;
7      #pragma omp parallel num_threads(num_threads) private(Aryabhata)
8      {
9          Aryabhata = 10;
10         int tid = omp_get_thread_num();
11
12         int result = tid * Aryabhata;
13
14         printf("Thread ID: %d, Result of Thread ID * Aryabhata: %d\n", tid, result);
15     }
16     return 0;
17 }
18
```

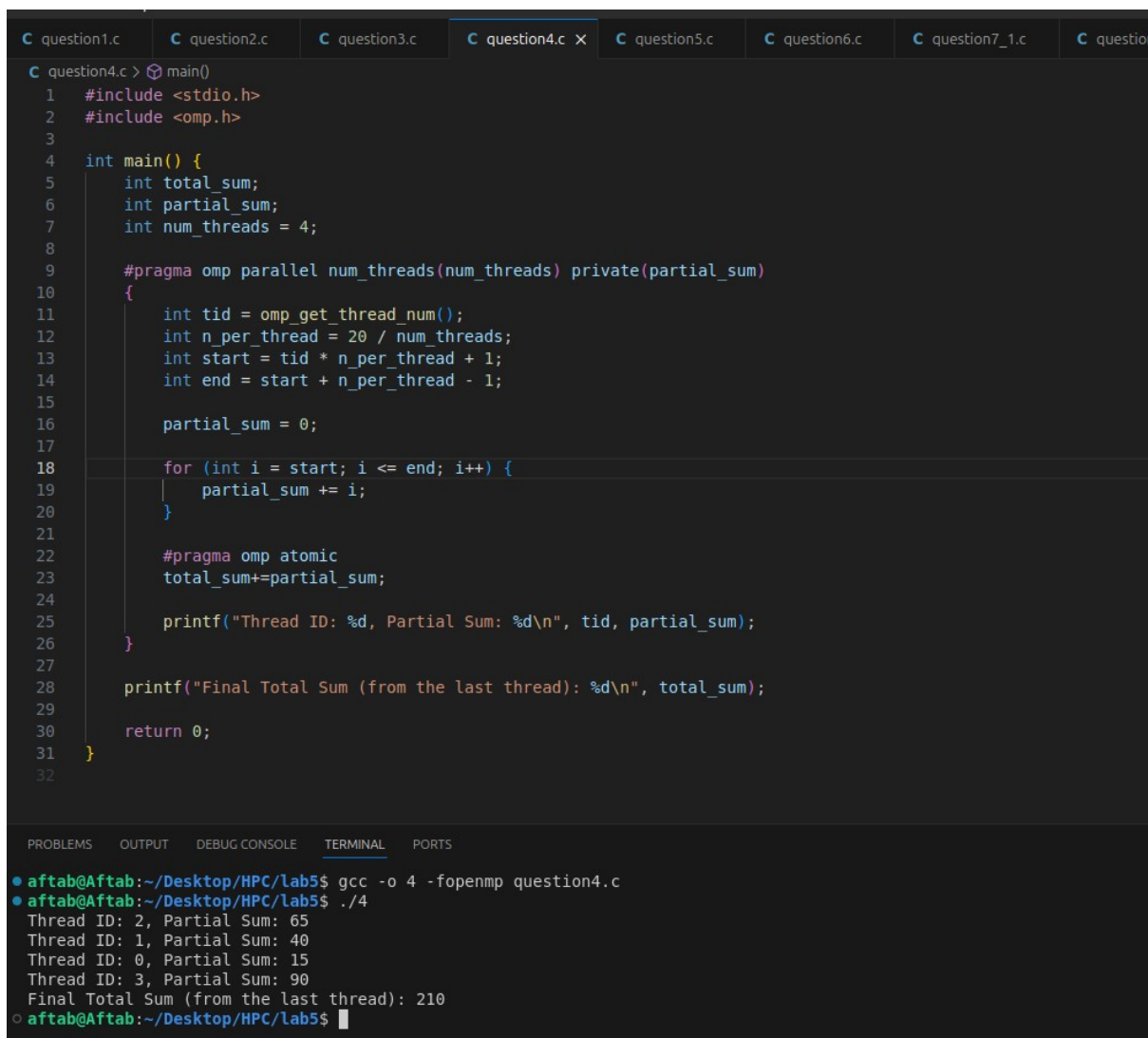
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/Lab5$ gcc -o 2 -fopenmp question2.c
● aftab@Aftab:~/Desktop/HPC/Lab5$ ./2
Thread ID: 0, Square: 0
Thread ID: 2, Square: 4
Thread ID: 1, Square: 1
Thread ID: 3, Square: 9
Sum of squares of thread IDs: 14
○ aftab@Aftab:~/Desktop/HPC/Lab5$
```

Q4. Write an OpenMP program that calculates the partial sum of the first 20 natural numbers using parallelism. Each thread should compute a portion of the sum by iterating through a loop. Implement the program using the lastprivate clause to ensure that the final total sum is correctly computed and printed outside the parallel region.

Hint:

- 1.Utilize OpenMP directives to parallelize the summation process.
- 2.Ensure that each thread has its private copy of partial sum.
- 3.Use the lastprivate clause to assign the value of the last thread's partial sum to the final total sum after the parallel region.



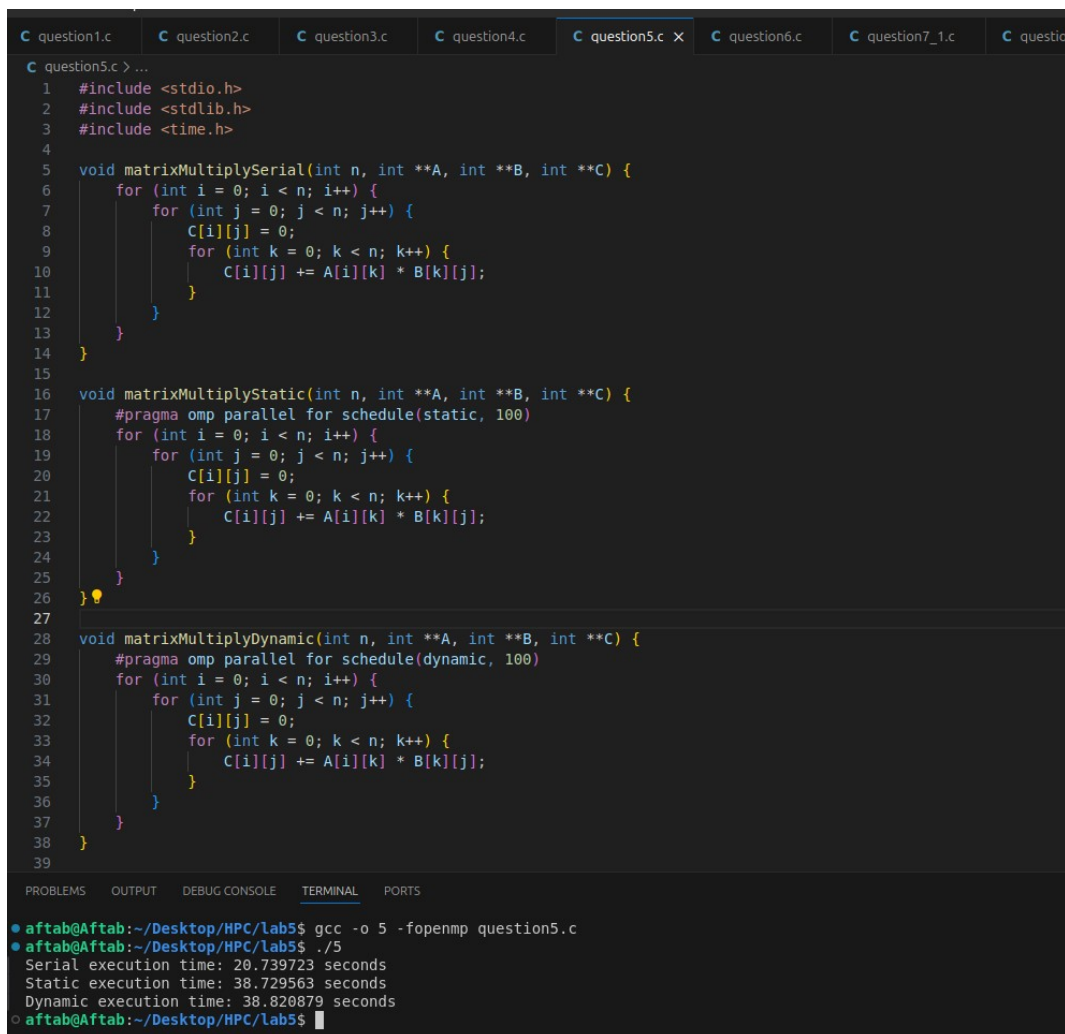
```
C question1.c C question2.c C question3.c C question4.c X C question5.c C question6.c C question7_1.c C question8.c
C question4.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main() {
5      int total_sum;
6      int partial_sum;
7      int num_threads = 4;
8
9      #pragma omp parallel num_threads(num_threads) private(partial_sum)
10     {
11         int tid = omp_get_thread_num();
12         int n_per_thread = 20 / num_threads;
13         int start = tid * n_per_thread + 1;
14         int end = start + n_per_thread - 1;
15
16         partial_sum = 0;
17
18         for (int i = start; i <= end; i++) {
19             partial_sum += i;
20         }
21
22         #pragma omp atomic
23         total_sum+=partial_sum;
24
25         printf("Thread ID: %d, Partial Sum: %d\n", tid, partial_sum);
26     }
27
28     printf("Final Total Sum (from the last thread): %d\n", total_sum);
29
30     return 0;
31 }
32

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● aftar@Aftar:~/Desktop/HPC/lab5$ gcc -o 4 -fopenmp question4.c
● aftar@Aftar:~/Desktop/HPC/lab5$ ./4
Thread ID: 2, Partial Sum: 65
Thread ID: 1, Partial Sum: 40
Thread ID: 0, Partial Sum: 15
Thread ID: 3, Partial Sum: 90
Final Total Sum (from the last thread): 210
○ aftar@Aftar:~/Desktop/HPC/lab5$
```

Q5. Consider a scenario where you have to parallelize a program that performs matrix multiplication using OpenMP. Your task is to implement parallelization using both static and dynamic scheduling, and compare the execution time of each approach.

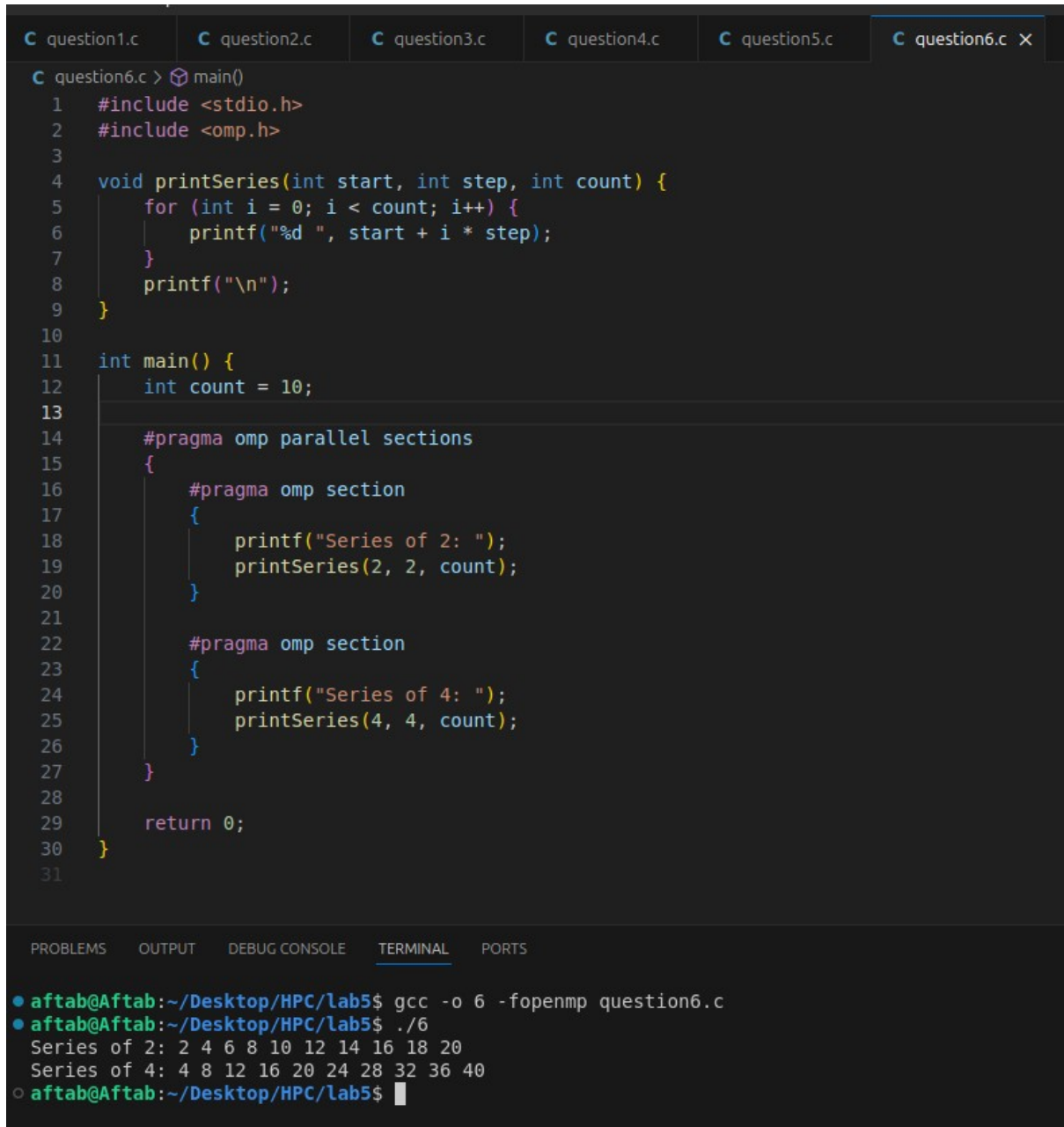
Note*:

1. Implement a serial version of matrix multiplication in C/C++.
2. Parallelize the matrix multiplication using OpenMP with static scheduling.
3. Parallelize the matrix multiplication using OpenMP with dynamic scheduling.
4. Measure the execution time of each parallelized version for various matrix sizes.
5. Compare the execution times and discuss the advantages and disadvantages of static and dynamic scheduling in this context.



```
C question1.c C question2.c C question3.c C question4.c C question5.c x C question6.c C question7_1.c C question8.c
C question5.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void matrixMultiplySerial(int n, int **A, int **B, int **C) {
6      for (int i = 0; i < n; i++) {
7          for (int j = 0; j < n; j++) {
8              C[i][j] = 0;
9              for (int k = 0; k < n; k++) {
10                 C[i][j] += A[i][k] * B[k][j];
11             }
12         }
13     }
14 }
15
16 void matrixMultiplyStatic(int n, int **A, int **B, int **C) {
17     #pragma omp parallel for schedule(static, 100)
18     for (int i = 0; i < n; i++) {
19         for (int j = 0; j < n; j++) {
20             C[i][j] = 0;
21             for (int k = 0; k < n; k++) {
22                 C[i][j] += A[i][k] * B[k][j];
23             }
24         }
25     }
26 }
27
28 void matrixMultiplyDynamic(int n, int **A, int **B, int **C) {
29     #pragma omp parallel for schedule(dynamic, 100)
30     for (int i = 0; i < n; i++) {
31         for (int j = 0; j < n; j++) {
32             C[i][j] = 0;
33             for (int k = 0; k < n; k++) {
34                 C[i][j] += A[i][k] * B[k][j];
35             }
36         }
37     }
38 }
39
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 5 -fopenmp question5.c
aftab@Aftab:~/Desktop/HPC/lab5$ ./5
Serial execution time: 20.739723 seconds
Static execution time: 38.729563 seconds
Dynamic execution time: 38.820879 seconds
aftab@Aftab:~/Desktop/HPC/lab5$
```

Q6. Write a Parallel C program which should print the series of 2 and 4. Make sure both should be executed by different threads !



```
C question1.c  C question2.c  C question3.c  C question4.c  C question5.c  C question6.c X
C question6.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  void printSeries(int start, int step, int count) {
5      for (int i = 0; i < count; i++) {
6          printf("%d ", start + i * step);
7      }
8      printf("\n");
9  }
10
11 int main() {
12     int count = 10;
13
14     #pragma omp parallel sections
15     {
16         #pragma omp section
17         {
18             printf("Series of 2: ");
19             printSeries(2, 2, count);
20         }
21
22         #pragma omp section
23         {
24             printf("Series of 4: ");
25             printSeries(4, 4, count);
26         }
27     }
28
29     return 0;
30 }
31
```

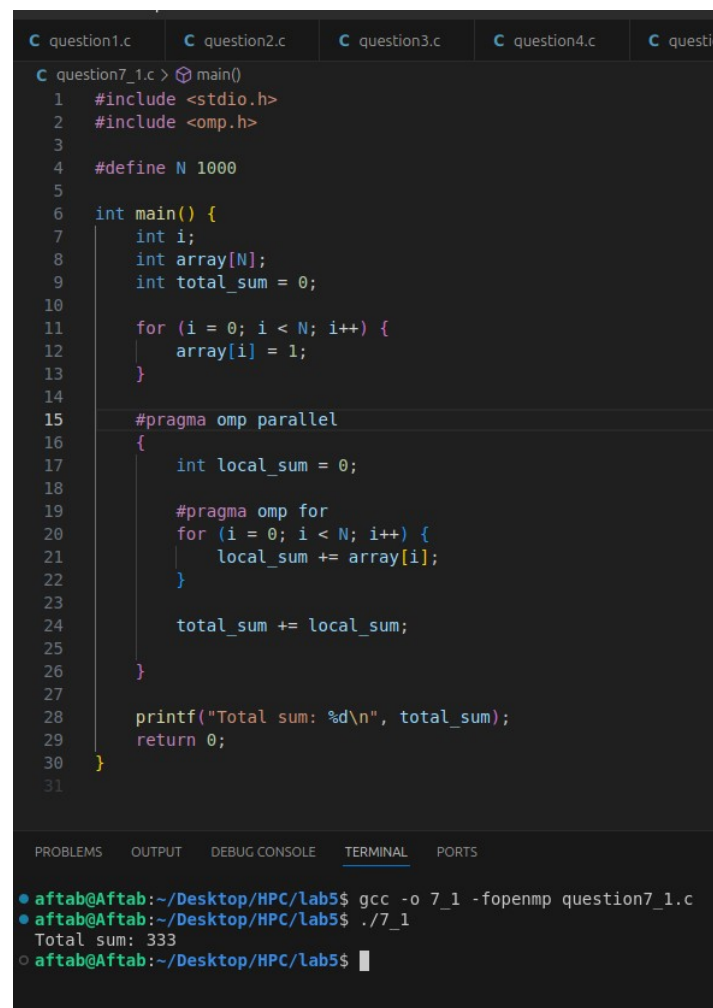
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 6 -fopenmp question6.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./6
Series of 2: 2 4 6 8 10 12 14 16 18 20
Series of 4: 4 8 12 16 20 24 28 32 36 40
○ aftab@Aftab:~/Desktop/HPC/lab5$
```

Q7. Consider a scenario where you have a shared variable `total_sum` that needs to be updated concurrently by multiple threads in a parallel program. However, concurrent updates to this variable can result in data races and incorrect results. Your task is to modify the program to ensure correct synchronization using OpenMP's critical and atomic constructs.

Note*:

- Implement a simple parallel program in C that initializes an array of integers and calculates the sum of its elements concurrently using OpenMP.
- Identify potential issues with concurrent updates to the `total_sum` variable in the parallelized version of the program.
- Modify the program to use OpenMP's critical/atomic directive to ensure synchronized access to the `total_sum` variable.
- Measure and compare the performance of synchronized versions against the unsynchronized implementation.



```
C question1.c C question2.c C question3.c C question4.c C question5.c
C question7_1.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 1000
5
6  int main() {
7      int i;
8      int array[N];
9      int total_sum = 0;
10
11     for (i = 0; i < N; i++) {
12         array[i] = 1;
13     }
14
15     #pragma omp parallel
16     {
17         int local_sum = 0;
18
19         #pragma omp for
20         for (i = 0; i < N; i++) {
21             local_sum += array[i];
22         }
23
24         total_sum += local_sum;
25     }
26
27     printf("Total sum: %d\n", total_sum);
28     return 0;
29 }
30
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/Lab5$ gcc -o 7_1 -fopenmp question7_1.c
● aftab@Aftab:~/Desktop/HPC/Lab5$ ./7_1
Total sum: 333
○ aftab@Aftab:~/Desktop/HPC/Lab5$
```


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

```
C question1.c C question2.c C question3.c C question4.c C question7_2.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 1000
5
6  int main() {
7      int i;
8      int array[N];
9      int total_sum = 0;
10
11     for (i = 0; i < N; i++) {
12         array[i] = 1;
13     }
14
15     #pragma omp parallel
16     {
17         int local_sum = 0;
18
19         #pragma omp for
20         for (i = 0; i < N; i++) {
21             local_sum += array[i];
22         }
23
24         #pragma omp critical
25         total_sum += local_sum;
26     }
27
28     printf("Total sum: %d\n", total_sum);
29     return 0;
30 }
31
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

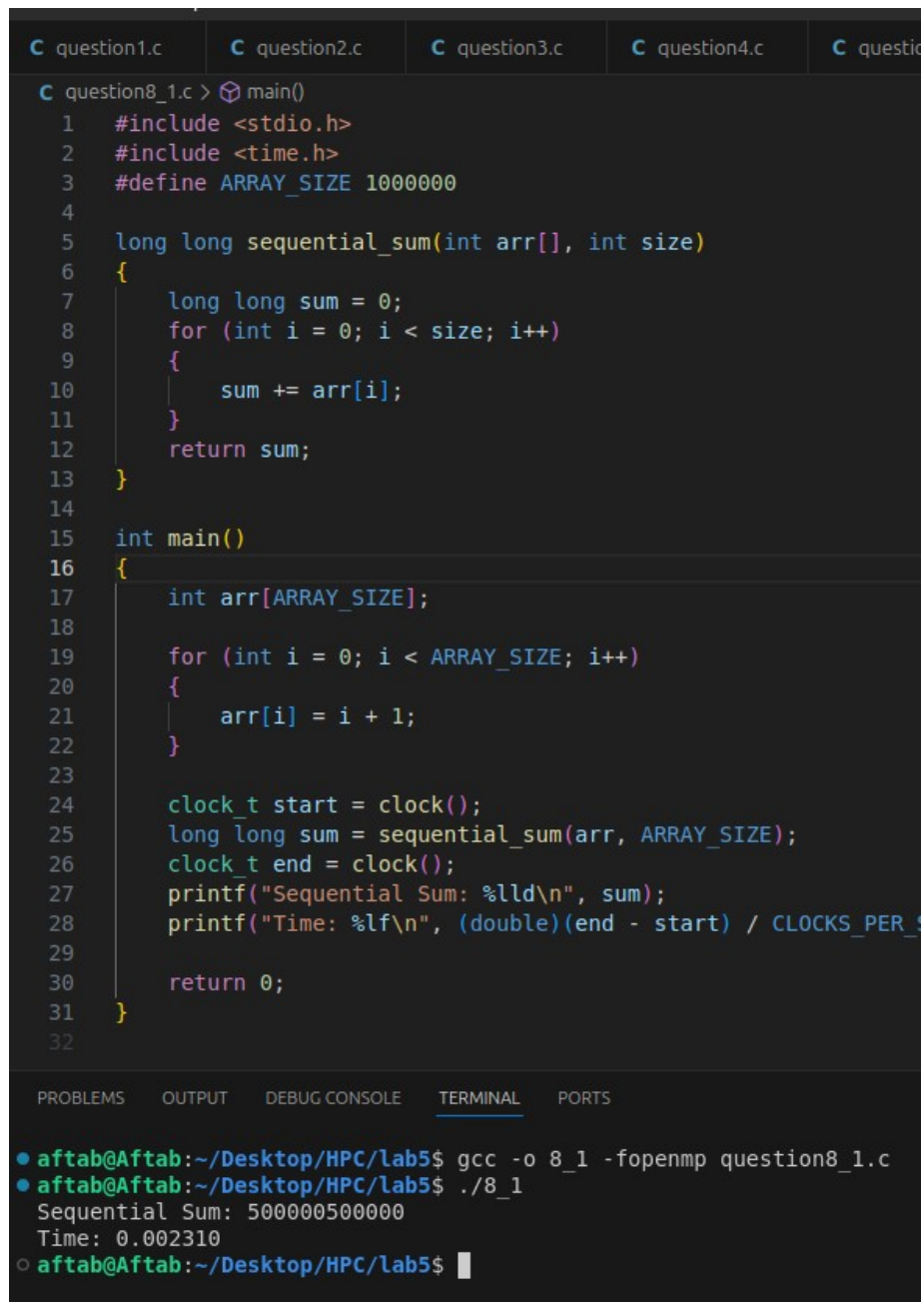
```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 7_2 -fopenmp question7_2.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./7_2
Total sum: 1000
○ aftab@Aftab:~/Desktop/HPC/lab5$
```

```
C question1.c C question2.c C question3.c C question4.c C question7_3.c > ...
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 1000
5
6  int main() {
7      int i;
8      int array[N];
9      int total_sum = 0;
10
11     for (i = 0; i < N; i++) {
12         array[i] = 1;
13     }
14
15     #pragma omp parallel
16     {
17         int local_sum = 0;
18
19         #pragma omp for
20         for (i = 0; i < N; i++) {
21             local_sum += array[i];
22         }
23
24         #pragma omp atomic
25         total_sum += local_sum;
26     }
27
28     printf("Total sum: %d\n", total_sum);
29     return 0;
30 }
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 7_3 -fopenmp question7_3.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./7_3
Total sum: 1000
○ aftab@Aftab:~/Desktop/HPC/lab5$
```


Q8. Consider a scenario where you have a large array of integers, and you need to find the sum of all its elements in parallel using OpenMP. The array is shared among multiple threads, and parallelism is needed to expedite the computation process. Your task is to write a parallel program that calculates the sum of all elements in the array using OpenMP's reduction clause.



```
C question1.c C question2.c C question3.c C question4.c C question5.c
C question8_1.c > main()
1  #include <stdio.h>
2  #include <time.h>
3  #define ARRAY_SIZE 1000000
4
5  long long sequential_sum(int arr[], int size)
6  {
7      long long sum = 0;
8      for (int i = 0; i < size; i++)
9      {
10         sum += arr[i];
11     }
12     return sum;
13 }
14
15 int main()
16 {
17     int arr[ARRAY_SIZE];
18
19     for (int i = 0; i < ARRAY_SIZE; i++)
20     {
21         arr[i] = i + 1;
22     }
23
24     clock_t start = clock();
25     long long sum = sequential_sum(arr, ARRAY_SIZE);
26     clock_t end = clock();
27     printf("Sequential Sum: %lld\n", sum);
28     printf("Time: %lf\n", (double)(end - start) / CLOCKS_PER_SEC);
29
30     return 0;
31 }
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 8_1 -fopenmp question8_1.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./8_1
Sequential Sum: 500000500000
Time: 0.002310
○ aftab@Aftab:~/Desktop/HPC/lab5$
```

```
C question1.c C question2.c C question3.c C question4.c C question8_2.c > parallel_sum(int [], int)
1  #include <stdio.h>
2  #include <omp.h>
3  #include <time.h>
4  #define ARRAY_SIZE 1000000
5
6  long long parallel_sum(int arr[], int size)
7  {
8      long long sum = 0;
9
10     #pragma omp parallel for reduction(+ : sum)
11     for (int i = 0; i < size; i++)
12     {
13         sum += arr[i];
14     }
15
16     return sum;
17 }
18
19 int main()
20 {
21     int arr[ARRAY_SIZE];
22
23     for (int i = 0; i < ARRAY_SIZE; i++)
24     {
25         arr[i] = i + 1;
26     }
27
28     clock_t start = clock();
29     long long sum = parallel_sum(arr, ARRAY_SIZE);
30     clock_t end = clock();
31     printf("Parallel Sum: %lld\n", sum);
32     printf("Time: %lf\n", (double)(end - start) / CLOCKS_PER_SEC);
33
34     return 0;
35 }
36
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/HPC/lab5$ gcc -o 8_2 -fopenmp question8_2.c
● aftab@Aftab:~/Desktop/HPC/lab5$ ./8_2
Parallel Sum: 500000500000
Time: 0.074876
○ aftab@Aftab:~/Desktop/HPC/lab5$
```