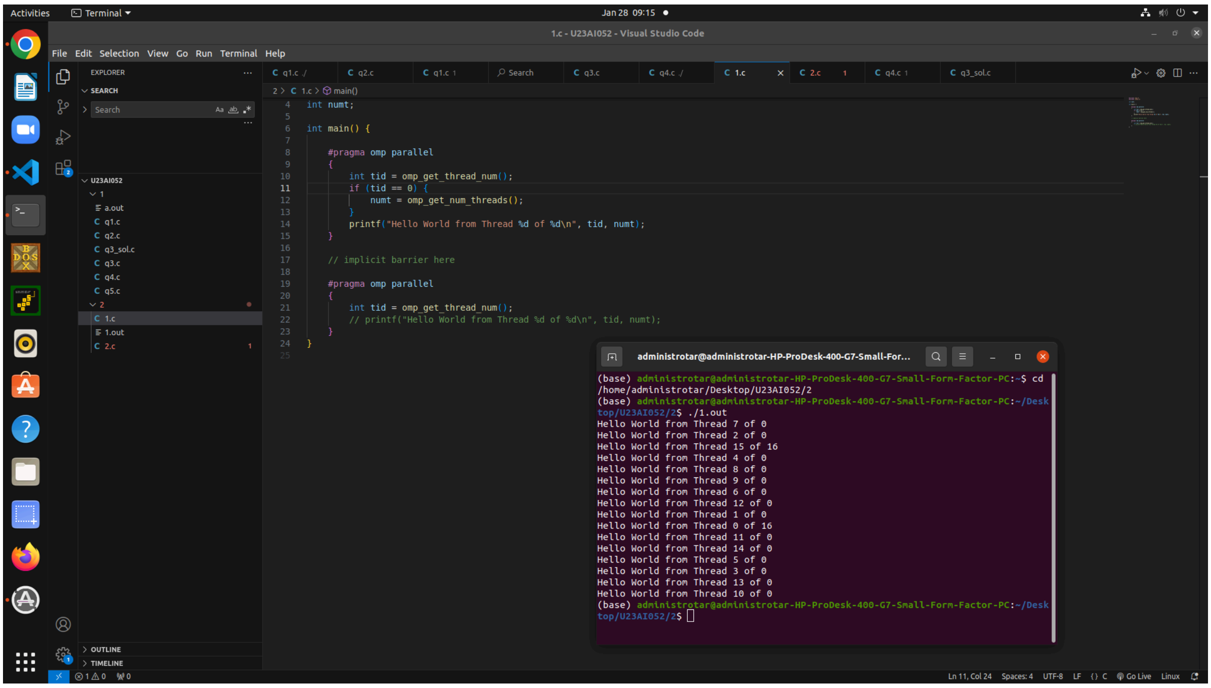
Lab Assignment: Analysing the parallel programming in openMP

Exercise 1: Synchronization Mechanisms in OpenMP

Objective: Write an OpenMP program to demonstrate various synchronization mechanisms. Your implementation should cover the following five approaches:

1. **Implicit Barriers:** Demonstrating how work-sharing constructs (like #pragma omp parallel) have a built-in barrier at the end. In fork – join block join block act as barrier.
2. **Implicit Barriers with Threadprivate Variables:** Using threadprivate data to show how persistence a particular variable retains its value across parallel regions interacts with synchronization.
3. **Explicit Barriers:** Implementing the #pragma omp barrier directive to manually synchronize all threads in a team.
4. **The single Directive:** Using #pragma omp single to ensure a block of code is executed by only one thread, with an implicit barrier for others. Also use this directive with nowait clause to see the execution of program.
5. **The master Directive:** Using #pragma omp master to execute code only on the master thread (noting the lack of an implicit barrier compared to single).

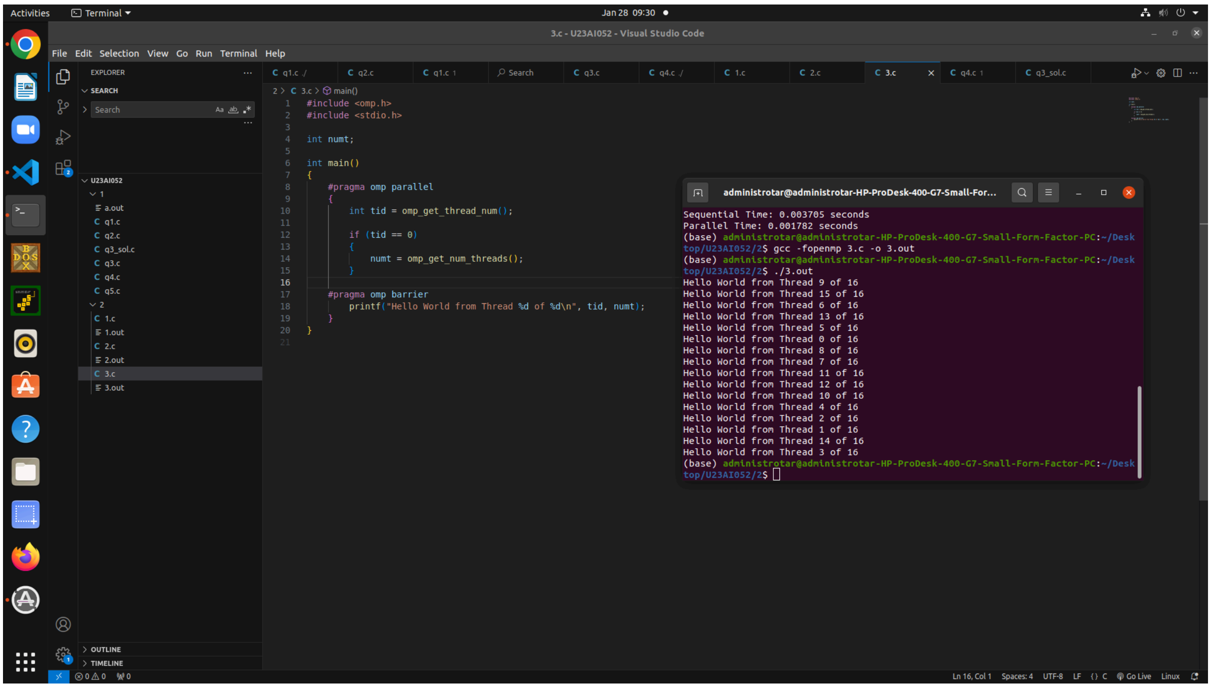


```c
int numt;

int main() {

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        if (tid == 0) {
            numt = omp_get_num_threads();
        }
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }

    // implicit barrier here

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        // printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~$ cd
/home/administrotar/Desktop/U23AI052/2
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./1.out
Hello World from Thread 7 of 0
Hello World from Thread 2 of 0
Hello World from Thread 15 of 16
Hello World from Thread 4 of 0
Hello World from Thread 8 of 0
Hello World from Thread 9 of 0
Hello World from Thread 6 of 0
Hello World from Thread 12 of 0
Hello World from Thread 1 of 0
Hello World from Thread 0 of 16
Hello World from Thread 11 of 0
Hello World from Thread 14 of 0
Hello World from Thread 5 of 0
Hello World from Thread 3 of 0
Hello World from Thread 13 of 0
Hello World from Thread 10 of 0
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ 
```

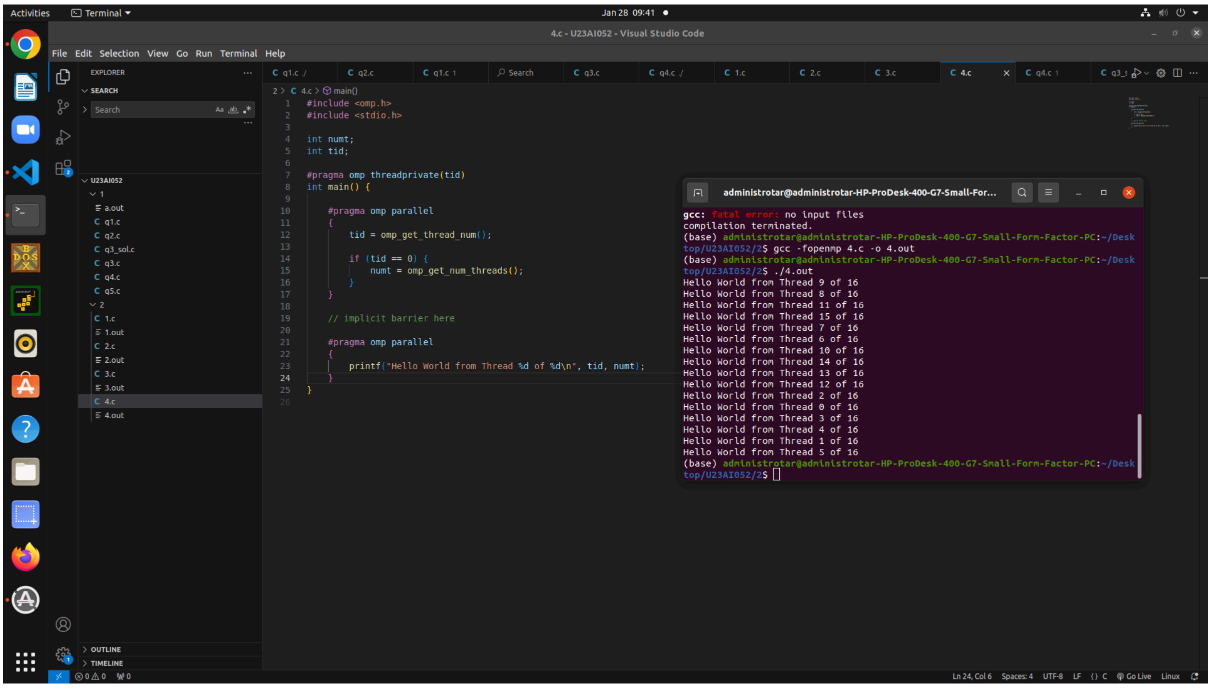
Activities
Terminal
Jan 28 09:15
1.c - U23AI052 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SEARCH
U23AI052
a.out
q1.c
q2.c
q3_sol.c
q3.c
q4.c
q5.c
1.c
1.out
2.c
OUTLINE
TIMELINE
q1.c ./
q2.c
q1.c 1
Search
q3.c
q4.c ./
1.c
2.c
q4.c 1
q3_sol.c
2 > 1.c > main()
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...
Ln 11, Col 24 Spaces: 4 UTF-8 LF {} C Go Live Linux

Analysis: We can see in the image below this that when i use 2 parallel blocks one after the other then openmp implicitly provides a barrier which causes all the threads in block 1 to complete their work first and then goes to the second one so it prints numt correctly as it is already calculated in parallel block 1 but in the image above this it is seen that it prints numt as 0 as when that thread was printed tid 0 was not generated and hence it gave us garbage value. Also this tells us that tid are not generated in numerical order.


Jan 28 09:30
3.c - U23AI052 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
2 > 3.c > main()
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...

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

int numt;

int main()
{
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        if (tid == 0)
        {
            numt = omp_get_num_threads();
        }

    #pragma omp barrier
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

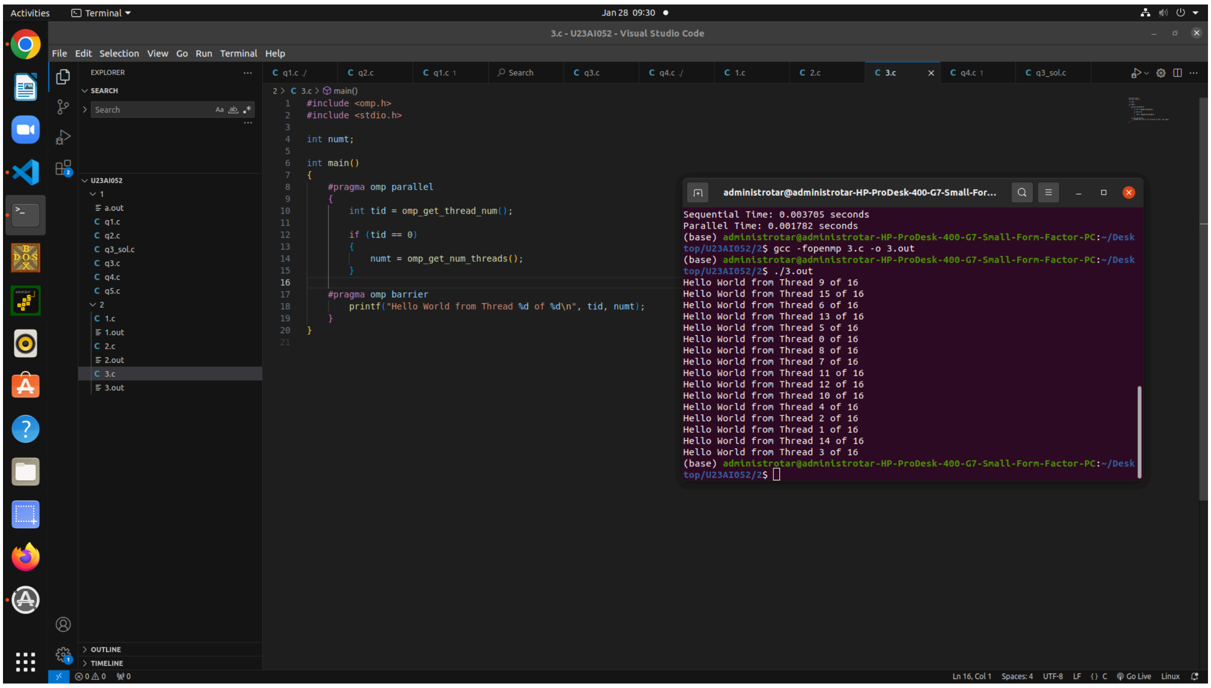
```
Sequential Time: 0.003705 seconds
Parallel Time: 0.001782 seconds
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 3.c -o 3.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./3.out
Hello World from Thread 9 of 16
Hello World from Thread 15 of 16
Hello World from Thread 6 of 16
Hello World from Thread 13 of 16
Hello World from Thread 5 of 16
Hello World from Thread 0 of 16
Hello World from Thread 8 of 16
Hello World from Thread 7 of 16
Hello World from Thread 11 of 16
Hello World from Thread 12 of 16
Hello World from Thread 10 of 16
Hello World from Thread 4 of 16
Hello World from Thread 2 of 16
Hello World from Thread 1 of 16
Hello World from Thread 14 of 16
Hello World from Thread 3 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ 
```


Jan 28 09:41
4.c - U23AI052 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
2 > 4.c > main()
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...

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

int numt;
int tid;

#pragma omp threadprivate(tid)
int main() {

    #pragma omp parallel
    {
        tid = omp_get_thread_num();

        if (tid == 0) {
            numt = omp_get_num_threads();
        }
    }

    // implicit barrier here

    #pragma omp parallel
    {
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
gcc: fatal error: no input files
compilation terminated.
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 4.c -o 4.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./4.out
Hello World from Thread 9 of 16
Hello World from Thread 8 of 16
Hello World from Thread 11 of 16
Hello World from Thread 15 of 16
Hello World from Thread 7 of 16
Hello World from Thread 6 of 16
Hello World from Thread 10 of 16
Hello World from Thread 14 of 16
Hello World from Thread 13 of 16
Hello World from Thread 12 of 16
Hello World from Thread 2 of 16
Hello World from Thread 0 of 16
Hello World from Thread 3 of 16
Hello World from Thread 4 of 16
Hello World from Thread 1 of 16
Hello World from Thread 5 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ 
```

Analysis: We can see here that tid 0 was generated at 12th position out of 16 and hence this confirms why 11 numbers in the above image showed tid as 0. Also we can see that threadprivate tid allowed a copy of tid in both the parallel block ultimately retaining its value across both parallel blocks.

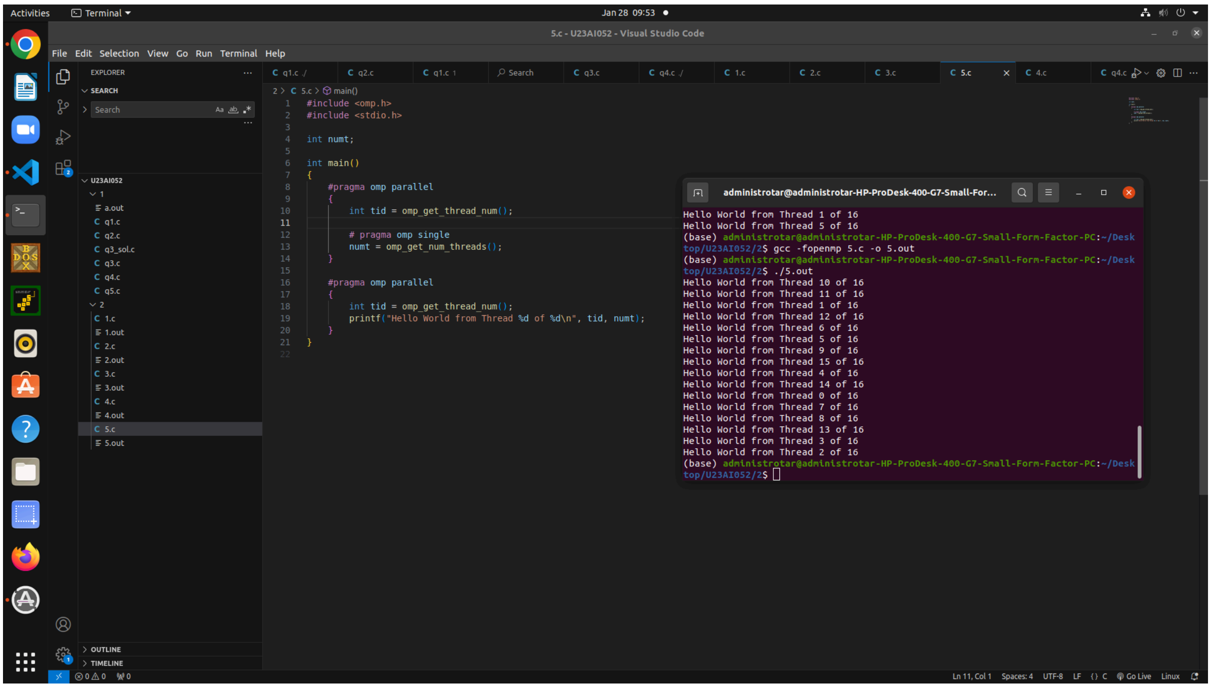


Activities
Terminal
Jan 28 09:30
3.c - U23AI052 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SEARCH
U23AI052
2 > C 3.c > main()
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...
OUTLINE
TIMELINE
Ln 16, Col 1 Spaces: 4 UTF-8 LF C Go Live Linux

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

int numt;

int main()
{
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        if (tid == 0)
        {
            numt = omp_get_num_threads();
        }

    #pragma omp barrier
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
Sequential Time: 0.003705 seconds
Parallel Time: 0.001782 seconds
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 3.c -o 3.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./3.out
Hello World from Thread 9 of 16
Hello World from Thread 15 of 16
Hello World from Thread 6 of 16
Hello World from Thread 13 of 16
Hello World from Thread 5 of 16
Hello World from Thread 0 of 16
Hello World from Thread 8 of 16
Hello World from Thread 7 of 16
Hello World from Thread 11 of 16
Hello World from Thread 12 of 16
Hello World from Thread 10 of 16
Hello World from Thread 4 of 16
Hello World from Thread 2 of 16
Hello World from Thread 1 of 16
Hello World from Thread 14 of 16
Hello World from Thread 3 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ 
```

Analysis: Here I have implemented the #pragma omp barrier preprocessor directive and by this inside a single parallel block also i am preventing the race condition that on the line number where i have written this directive, before it all the threads should complete the work which is to be done before that line and then only the code goes ahead, so it makes all the threads wait till all comes at the directive line and then they proceed ahead.

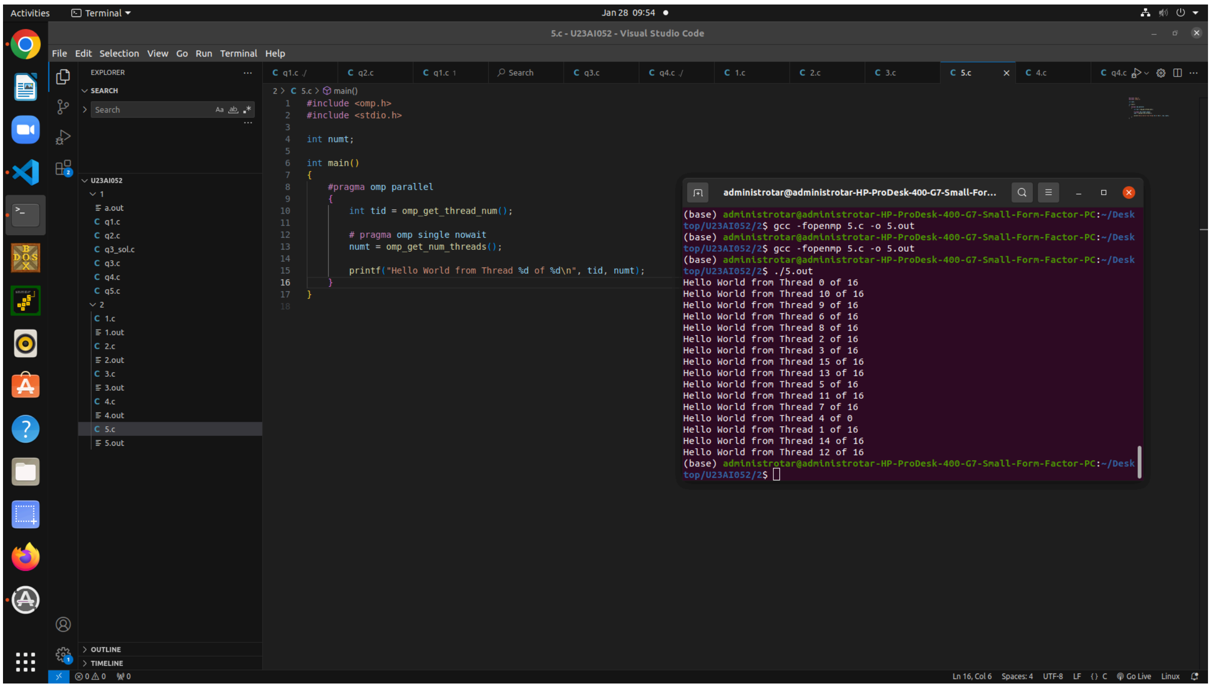


Activities
Terminal
Jan 28 09:53
5.c - U23AI052 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SEARCH
U23AI052
2 > C 5.c > main()
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...
OUTLINE
TIMELINE
Ln 11, Col 1 Spaces: 4 UTF-8 LF C Go Live Linux

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

int numt;

int main()
{
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        # pragma omp single
        numt = omp_get_num_threads();
    }

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
Hello World from Thread 1 of 16
Hello World from Thread 5 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 5.c -o 5.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./5.out
Hello World from Thread 10 of 16
Hello World from Thread 11 of 16
Hello World from Thread 1 of 16
Hello World from Thread 12 of 16
Hello World from Thread 6 of 16
Hello World from Thread 5 of 16
Hello World from Thread 9 of 16
Hello World from Thread 15 of 16
Hello World from Thread 4 of 16
Hello World from Thread 14 of 16
Hello World from Thread 0 of 16
Hello World from Thread 7 of 16
Hello World from Thread 8 of 16
Hello World from Thread 13 of 16
Hello World from Thread 3 of 16
Hello World from Thread 2 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ 
```

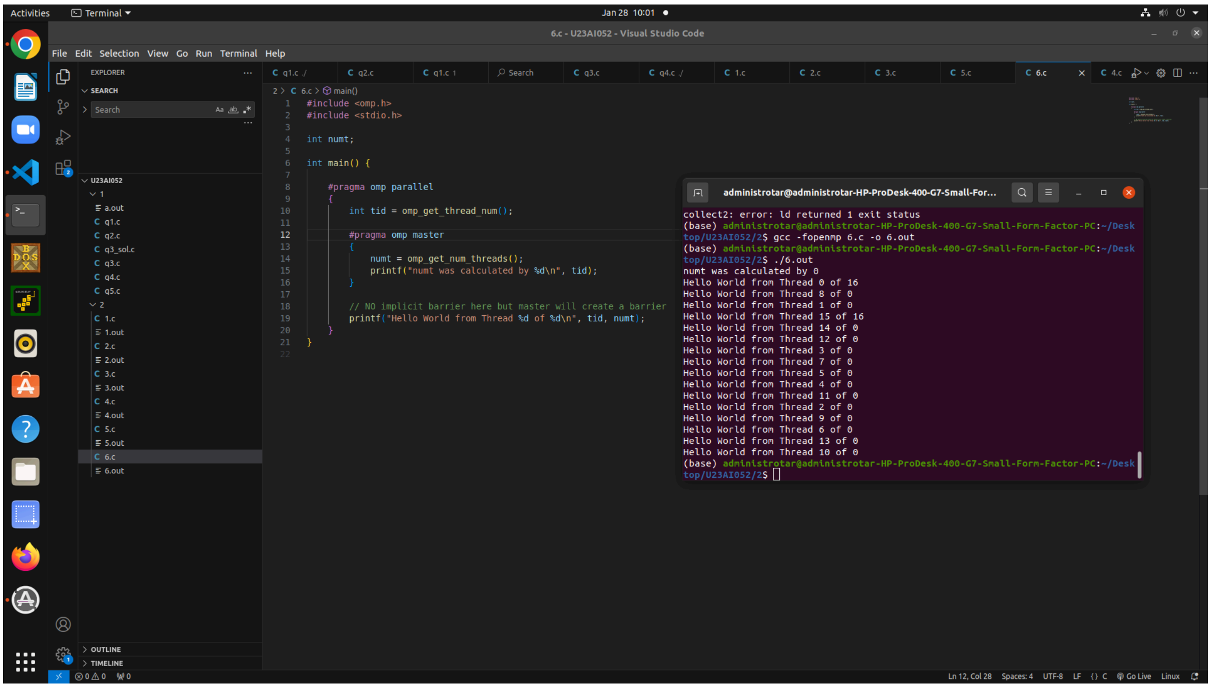
Analysis: This shows that if we use # pragma omp single it allows only one thread to execute the numt calculation and by default it waits at this line for all the threads to come and this single thread to complete its execution



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

int numt;

int main()
{
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        # pragma omp single nowait
        numt = omp_get_num_threads();

        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 5.c -o 5.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 5.c -o 5.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./5.out
Hello World from Thread 0 of 16
Hello World from Thread 10 of 16
Hello World from Thread 9 of 16
Hello World from Thread 6 of 16
Hello World from Thread 8 of 16
Hello World from Thread 2 of 16
Hello World from Thread 3 of 16
Hello World from Thread 15 of 16
Hello World from Thread 13 of 16
Hello World from Thread 5 of 16
Hello World from Thread 11 of 16
Hello World from Thread 7 of 16
Hello World from Thread 4 of 0
Hello World from Thread 1 of 16
Hello World from Thread 14 of 16
Hello World from Thread 12 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$
```

Analysis: This shows that if we use # pragma omp single nowait it allows only one thread to execute the numt calculation but it doesnt wait this time at this line for all the threads to come and allows others to go and print the hello world thing.



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

int numt;

int main() {

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();

        #pragma omp master
        {
            numt = omp_get_num_threads();
            printf("numt was calculated by %d\n", tid);
        }

        // NO implicit barrier here but master will create a barrier
        printf("Hello World from Thread %d of %d\n", tid, numt);
    }
}
```

```
collect2: error: ld returned 1 exit status
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 6.c -o 6.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./6.out
numt was calculated by 0
Hello World from Thread 0 of 16
Hello World from Thread 8 of 0
Hello World from Thread 1 of 0
Hello World from Thread 15 of 16
Hello World from Thread 14 of 0
Hello World from Thread 12 of 0
Hello World from Thread 3 of 0
Hello World from Thread 7 of 0
Hello World from Thread 5 of 0
Hello World from Thread 4 of 0
Hello World from Thread 11 of 0
Hello World from Thread 2 of 0
Hello World from Thread 9 of 0
Hello World from Thread 6 of 0
Hello World from Thread 13 of 0
Hello World from Thread 10 of 0
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$
```

Analysis: Here i used # pragma omp master, i also tried to print the tid of the thread that calculated tid and yes it verified that master thread is always 0 as i ran it 4-5 times and all the times it was tid 0 that was master thread.

One more interesting thing to notice is that other threads don’t wait for master to complete its work and hence in some cases i got the garbage value as master thread 0 was not yet created. This shows lack of implicit barrier in master thread.

Exercise 2: Performance Analysis of Array Summation

Objective: Develop a program to compute the sum of an array and measure the execution time (latency) for each of the following scenarios:

Part A: Sequential Implementation

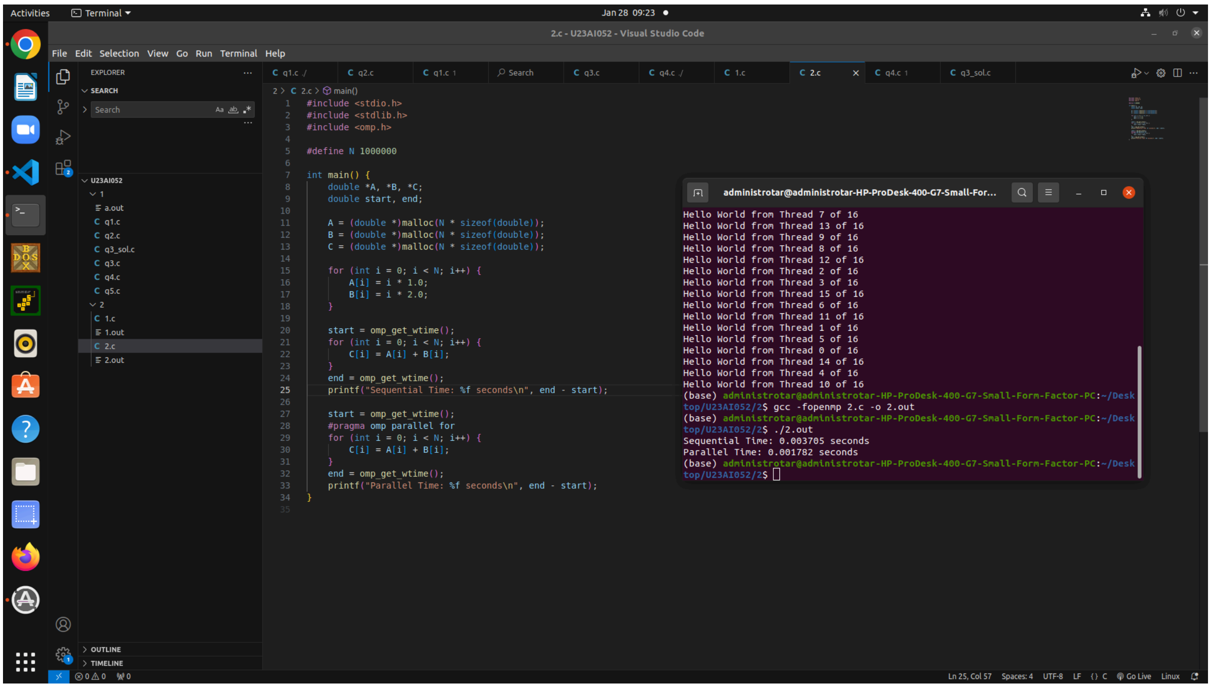
- Compute the sum of the array using a standard serial approach to establish a performance baseline.

Part B: Parallel Implementation

Implement the parallel sum using two different synchronization strategies to compare their efficiency:

1. **Critical Section:** Use the #pragma omp critical directive to protect the global sum variable from race conditions.
2. **Reduction Operator:** Use the reduction(+:sum) clause to allow OpenMP to manage thread-local sums and perform an efficient final merge.

Use omp_get_wtime() function to call the execution time of a program for particular block.



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

#define N 1000000

int main() {
    double *A, *B, *C;
    double start, end;

    A = (double *)malloc(N * sizeof(double));
    B = (double *)malloc(N * sizeof(double));
    C = (double *)malloc(N * sizeof(double));

    for (int i = 0; i < N; i++) {
        A[i] = i * 1.0;
        B[i] = i * 2.0;
    }

    start = omp_get_wtime();
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
    }
    end = omp_get_wtime();
    printf("Sequential Time: %f seconds\n", end - start);

    start = omp_get_wtime();
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
    }
    end = omp_get_wtime();
    printf("Parallel Time: %f seconds\n", end - start);
}
```

```
Hello World from Thread 7 of 16
Hello World from Thread 13 of 16
Hello World from Thread 9 of 16
Hello World from Thread 8 of 16
Hello World from Thread 12 of 16
Hello World from Thread 2 of 16
Hello World from Thread 3 of 16
Hello World from Thread 15 of 16
Hello World from Thread 6 of 16
Hello World from Thread 11 of 16
Hello World from Thread 1 of 16
Hello World from Thread 5 of 16
Hello World from Thread 0 of 16
Hello World from Thread 14 of 16
Hello World from Thread 4 of 16
Hello World from Thread 10 of 16
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ gcc -fopenmp 2.c -o 2.out
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$ ./2.out
Sequential Time: 0.003705 seconds
Parallel Time: 0.001782 seconds
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052/2$
```

Analysis: Here we can see that the time for the parallel execution is 0.001782 seconds and that for sequential execution is 0.003705 seconds so we can confirm parallel execution helps in reducing the time which is one of the main purpose of using openmp.