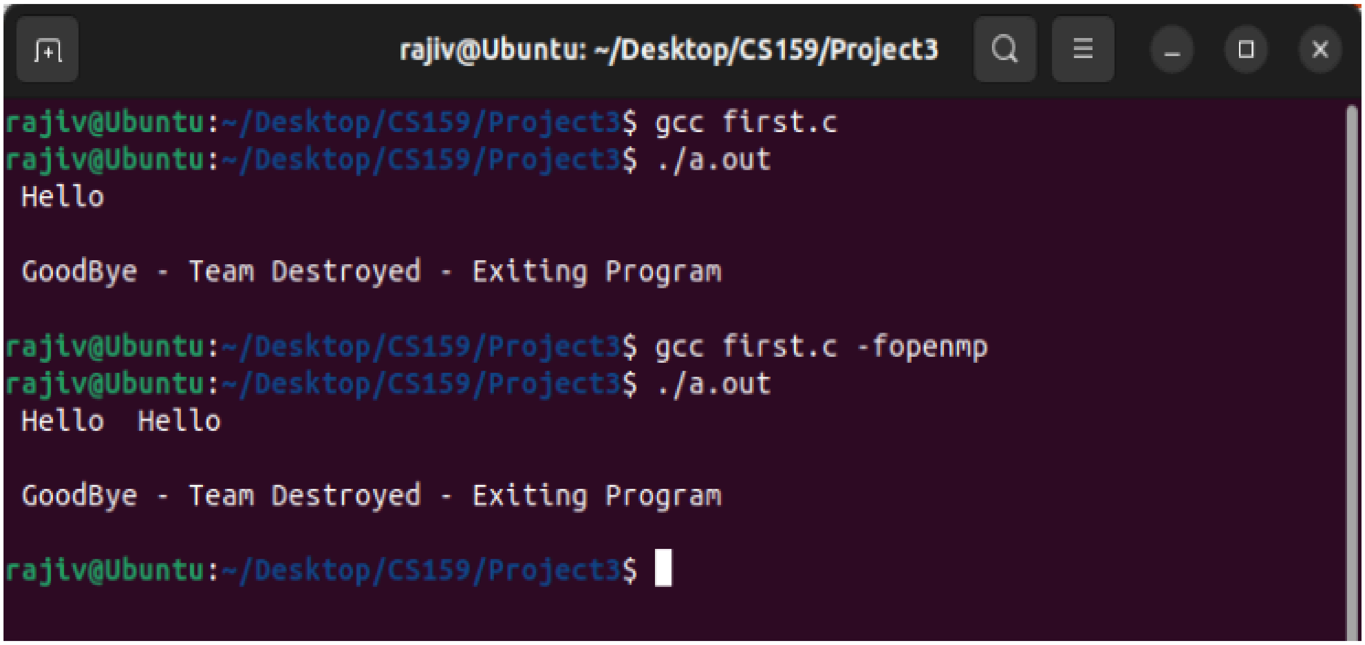
CS159 - Project 3
Rajiv Karthik Reddy Kodimala
Student ID: 017415801

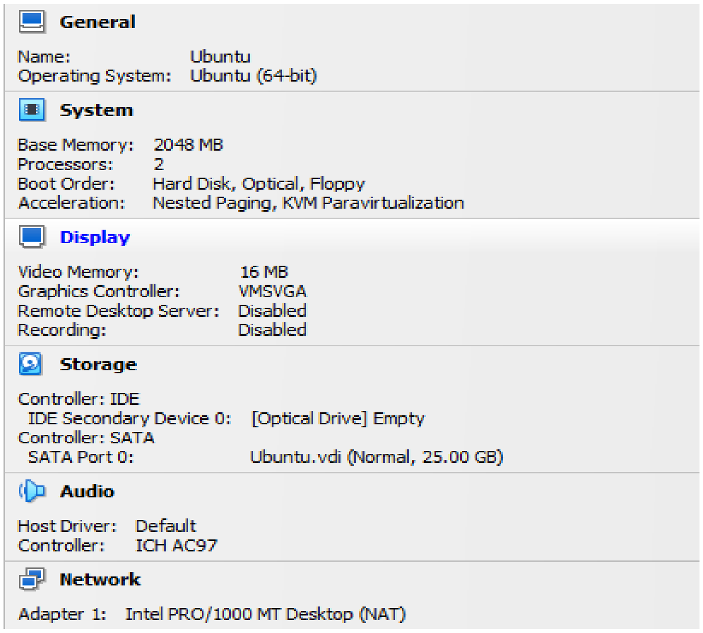
1.



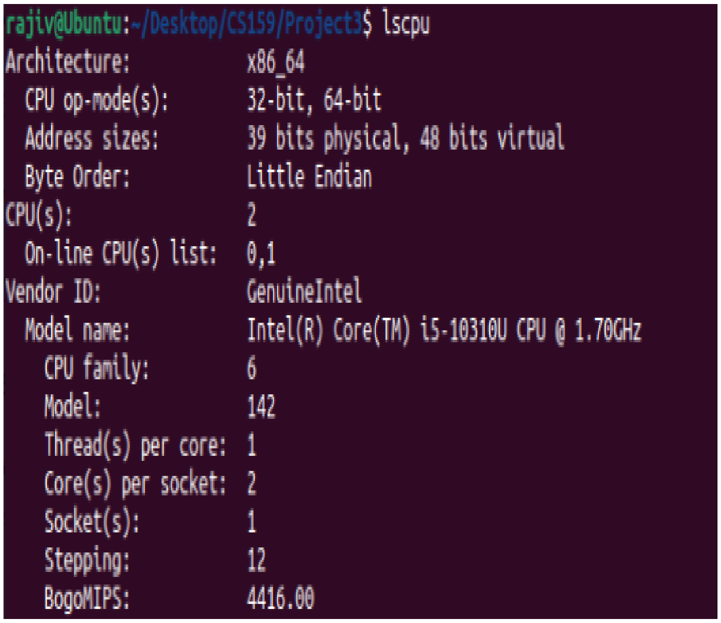
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc first.c
rajiv@Ubuntu:~/Desktop/CS159/Project3$ ./a.out
 Hello

 GoodBye - Team Destroyed - Exiting Program

rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc first.c -fopenmp
rajiv@Ubuntu:~/Desktop/CS159/Project3$ ./a.out
 Hello  Hello

 GoodBye - Team Destroyed - Exiting Program

rajiv@Ubuntu:~/Desktop/CS159/Project3$ 
```

Observations: When the -fopenmp flag is not added, the program only outputs a single Hello message. With the -fopenmp flag, the program outputs 2 Hello messages. This indicates without -fopenmp flag, pragma was ignored, whereas with -fopenmp, 2 threads were running in parallel.

Note: Program was run in a VM machine with 2 cores and below are the VM details.

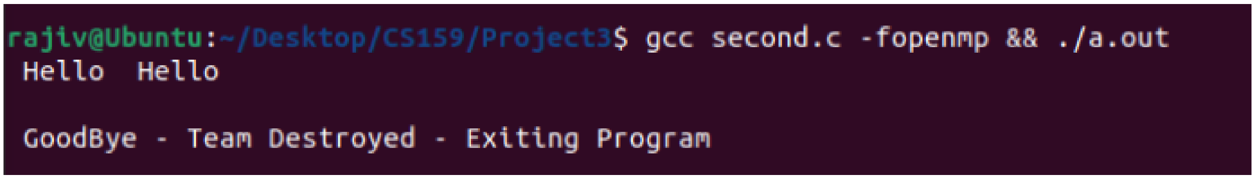


```
General
Name:               Ubuntu
Operating System:   Ubuntu (64-bit)

System
Base Memory:    2048 MB
Processors:     2
Boot Order:     Hard Disk, Optical, Floppy
Acceleration:   Nested Paging, KVM Paravirtualization

Display
Video Memory:             16 MB
Graphics Controller:      VMSVGA
Remote Desktop Server:    Disabled
Recording:                Disabled

Storage
Controller: IDE
  IDE Secondary Device 0:   [Optical Drive] Empty
Controller: SATA
  SATA Port 0:              Ubuntu.vdi (Normal, 25.00 GB)

Audio
Host Driver:   Default
Controller:    ICH AC97

Network
Adapter 1:   Intel PRO/1000 MT Desktop (NAT)
```



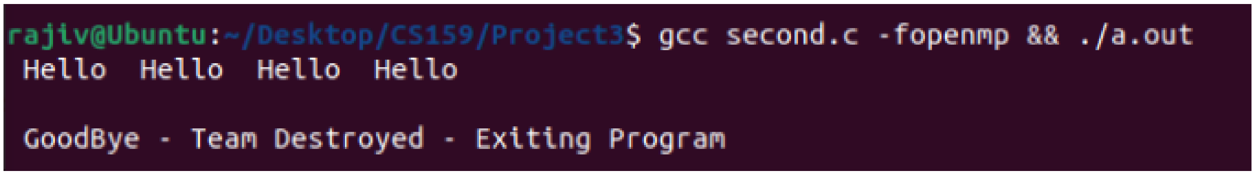
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  2
  On-line CPU(s) list:   0,1
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i5-10310U CPU @ 1.70GHz
    CPU family:          6
    Model:               142
    Thread(s) per core:  1
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            12
    BogoMIPS:            4416.00
```

2. With 2 threads:



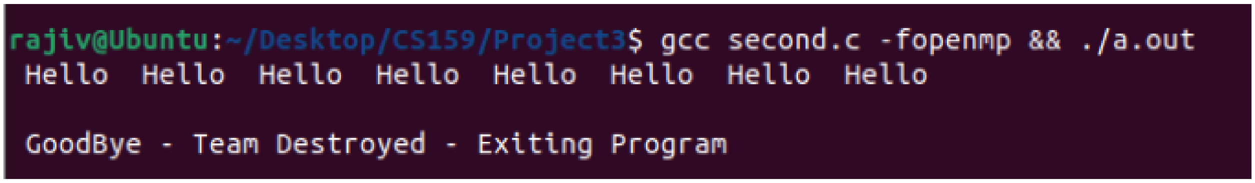
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc second.c -fopenmp && ./a.out
 Hello  Hello

 GoodBye - Team Destroyed - Exiting Program
```

With 4 threads:



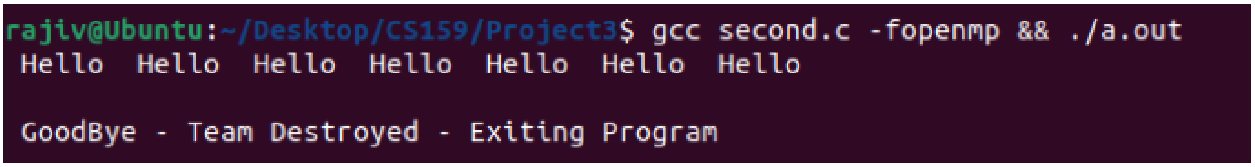
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc second.c -fopenmp && ./a.out
 Hello  Hello  Hello  Hello

 GoodBye - Team Destroyed - Exiting Program
```

With 8 threads: (more than number of cores)



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc second.c -fopenmp && ./a.out
 Hello  Hello  Hello  Hello  Hello  Hello  Hello  Hello

 GoodBye - Team Destroyed - Exiting Program
```

With 7 threads:

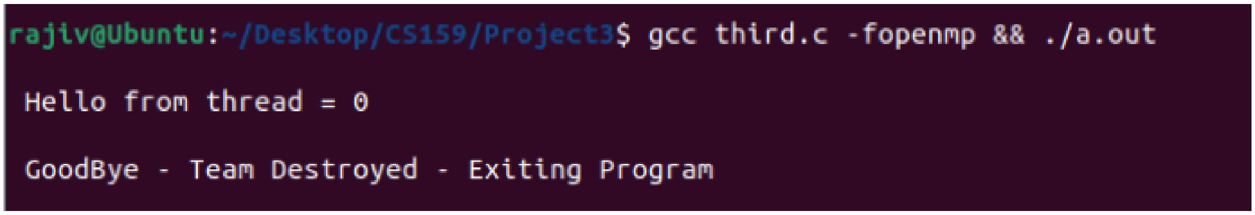


```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc second.c -fopenmp && ./a.out
 Hello  Hello  Hello  Hello  Hello  Hello  Hello

 GoodBye - Team Destroyed - Exiting Program
```

As we can see from the results, The program runs fine with 2, 4, or 8 threads. It can also work with an odd number of threads or more threads than there are CPU cores.

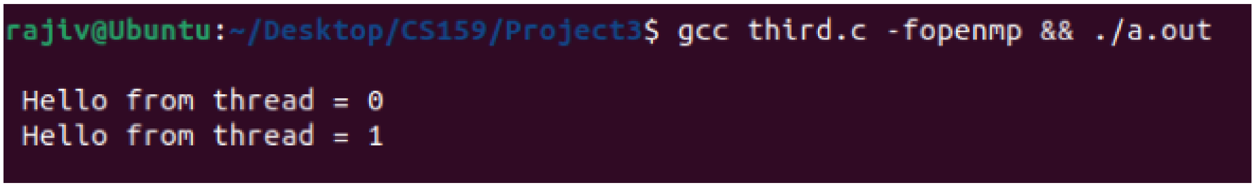
3.

With 1 thread:



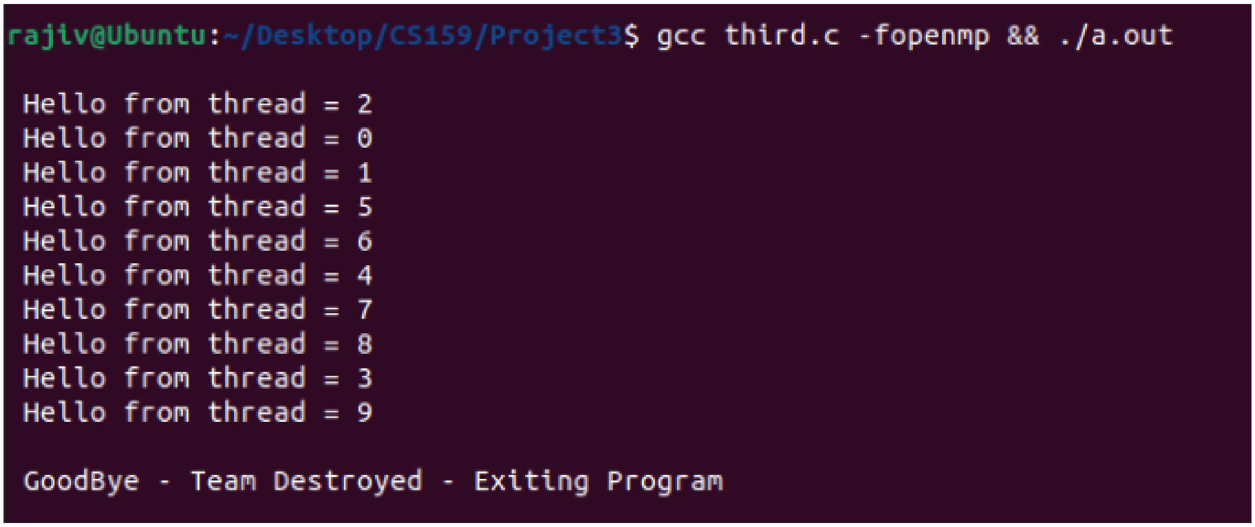
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc third.c -fopenmp && ./a.out

 Hello from thread = 0

 GoodBye - Team Destroyed - Exiting Program
```

With 2 threads:



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc third.c -fopenmp && ./a.out

 Hello from thread = 0
 Hello from thread = 1
```

With 10 threads:

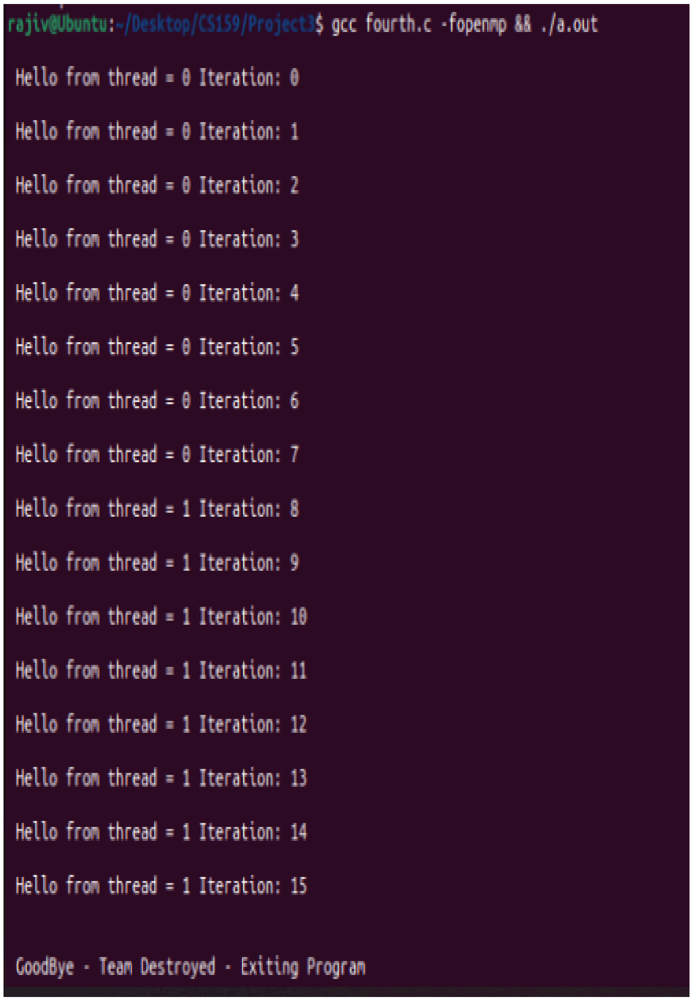


```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc third.c -fopenmp && ./a.out

 Hello from thread = 2
 Hello from thread = 0
 Hello from thread = 1
 Hello from thread = 5
 Hello from thread = 6
 Hello from thread = 4
 Hello from thread = 7
 Hello from thread = 8
 Hello from thread = 3
 Hello from thread = 9

 GoodBye - Team Destroyed - Exiting Program
```

The output above clearly shows that the threads operate concurrently and don't necessarily follow the order in which they were created.

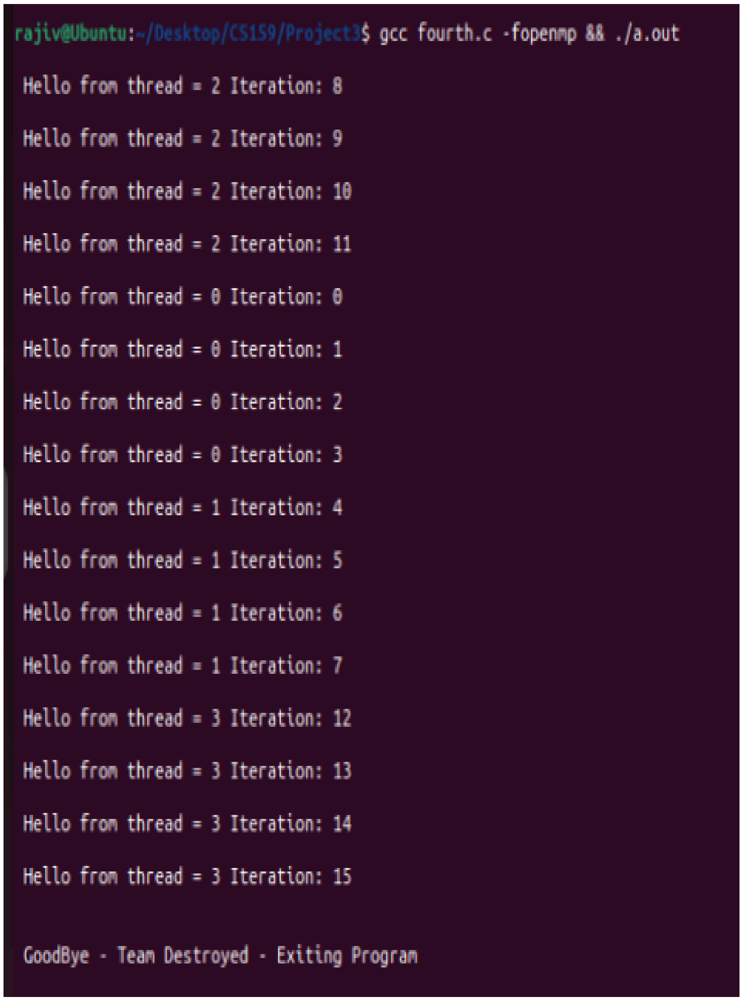
4.

With 2 threads:



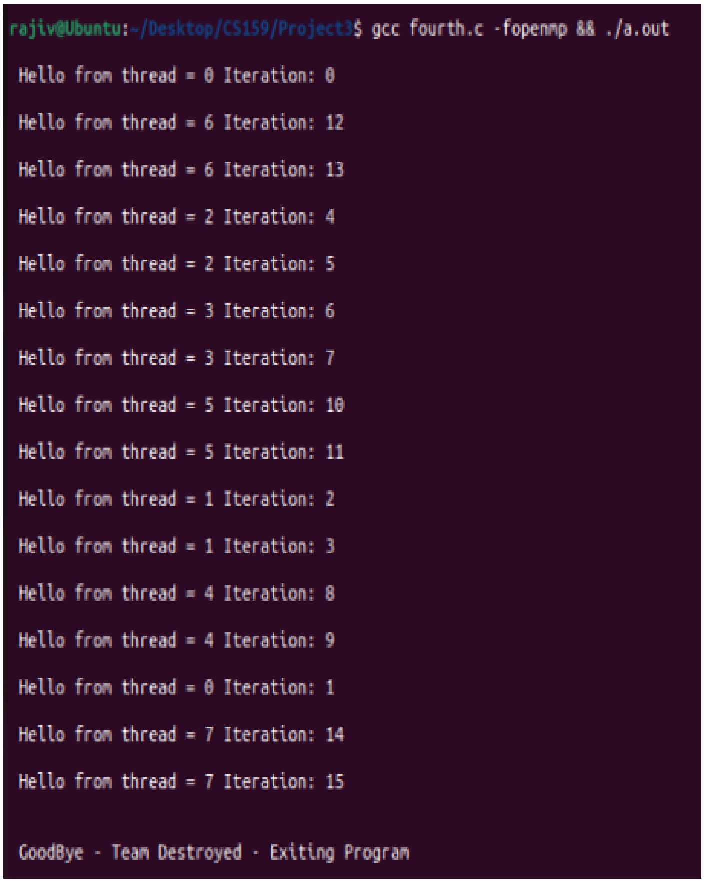
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 0 Iteration: 0

 Hello from thread = 0 Iteration: 1

 Hello from thread = 0 Iteration: 2

 Hello from thread = 0 Iteration: 3

 Hello from thread = 0 Iteration: 4

 Hello from thread = 0 Iteration: 5

 Hello from thread = 0 Iteration: 6

 Hello from thread = 0 Iteration: 7

 Hello from thread = 1 Iteration: 8

 Hello from thread = 1 Iteration: 9

 Hello from thread = 1 Iteration: 10

 Hello from thread = 1 Iteration: 11

 Hello from thread = 1 Iteration: 12

 Hello from thread = 1 Iteration: 13

 Hello from thread = 1 Iteration: 14

 Hello from thread = 1 Iteration: 15


 GoodBye - Team Destroyed - Exiting Program
```

With 4 threads:



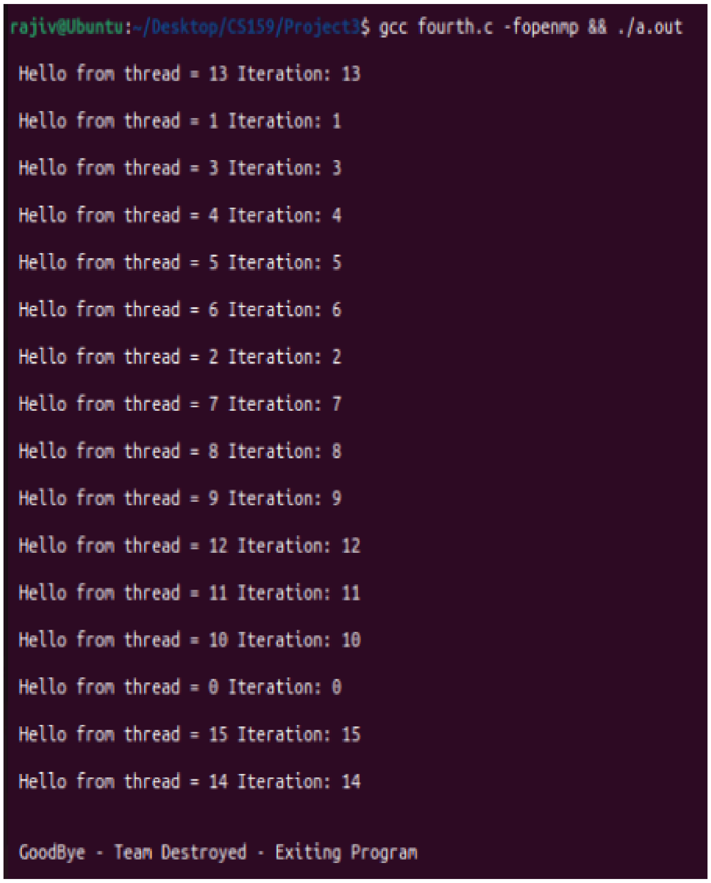
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 2 Iteration: 8

 Hello from thread = 2 Iteration: 9

 Hello from thread = 2 Iteration: 10

 Hello from thread = 2 Iteration: 11

 Hello from thread = 0 Iteration: 0

 Hello from thread = 0 Iteration: 1

 Hello from thread = 0 Iteration: 2

 Hello from thread = 0 Iteration: 3

 Hello from thread = 1 Iteration: 4

 Hello from thread = 1 Iteration: 5

 Hello from thread = 1 Iteration: 6

 Hello from thread = 1 Iteration: 7

 Hello from thread = 3 Iteration: 12

 Hello from thread = 3 Iteration: 13

 Hello from thread = 3 Iteration: 14

 Hello from thread = 3 Iteration: 15


 GoodBye - Team Destroyed - Exiting Program
```

With 8 threads:



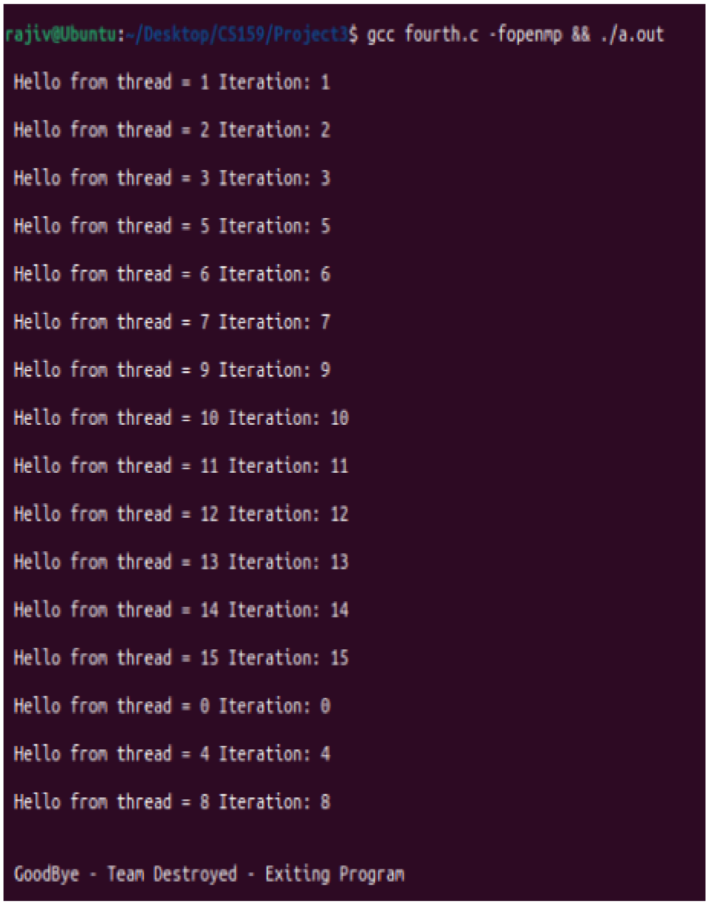
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 0 Iteration: 0

 Hello from thread = 6 Iteration: 12

 Hello from thread = 6 Iteration: 13

 Hello from thread = 2 Iteration: 4

 Hello from thread = 2 Iteration: 5

 Hello from thread = 3 Iteration: 6

 Hello from thread = 3 Iteration: 7

 Hello from thread = 5 Iteration: 10

 Hello from thread = 5 Iteration: 11

 Hello from thread = 1 Iteration: 2

 Hello from thread = 1 Iteration: 3

 Hello from thread = 4 Iteration: 8

 Hello from thread = 4 Iteration: 9

 Hello from thread = 0 Iteration: 1

 Hello from thread = 7 Iteration: 14

 Hello from thread = 7 Iteration: 15


 GoodBye - Team Destroyed - Exiting Program
```

With 16 threads:



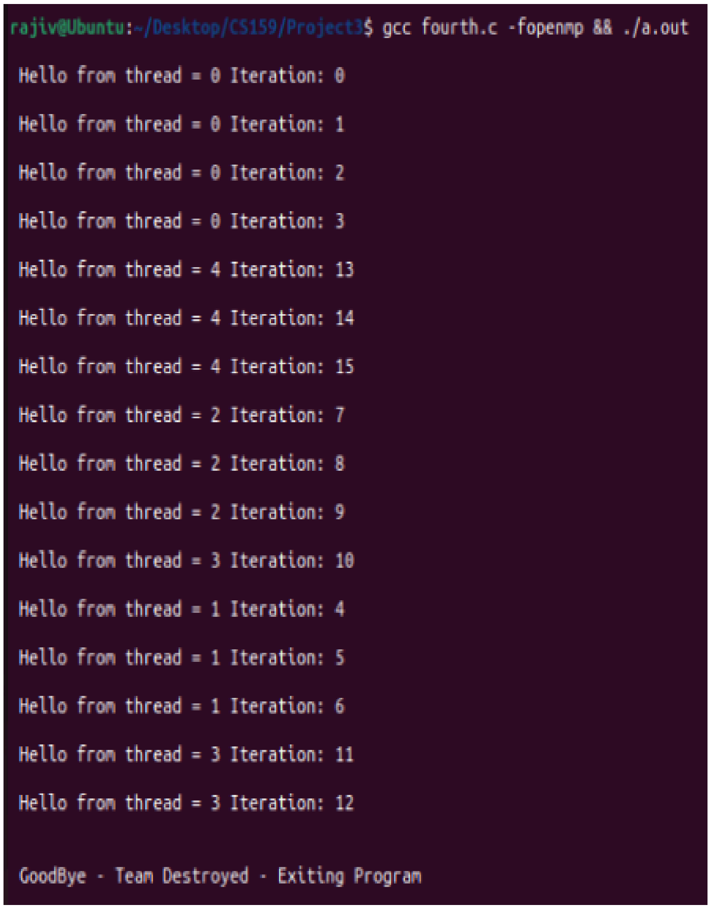
```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 13 Iteration: 13

 Hello from thread = 1 Iteration: 1

 Hello from thread = 3 Iteration: 3

 Hello from thread = 4 Iteration: 4

 Hello from thread = 5 Iteration: 5

 Hello from thread = 6 Iteration: 6

 Hello from thread = 2 Iteration: 2

 Hello from thread = 7 Iteration: 7

 Hello from thread = 8 Iteration: 8

 Hello from thread = 9 Iteration: 9

 Hello from thread = 12 Iteration: 12

 Hello from thread = 11 Iteration: 11

 Hello from thread = 10 Iteration: 10

 Hello from thread = 0 Iteration: 0

 Hello from thread = 15 Iteration: 15

 Hello from thread = 14 Iteration: 14


 GoodBye - Team Destroyed - Exiting Program
```

With 32 threads:



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 1 Iteration: 1

 Hello from thread = 2 Iteration: 2

 Hello from thread = 3 Iteration: 3

 Hello from thread = 5 Iteration: 5

 Hello from thread = 6 Iteration: 6

 Hello from thread = 7 Iteration: 7

 Hello from thread = 9 Iteration: 9

 Hello from thread = 10 Iteration: 10

 Hello from thread = 11 Iteration: 11

 Hello from thread = 12 Iteration: 12

 Hello from thread = 13 Iteration: 13

 Hello from thread = 14 Iteration: 14

 Hello from thread = 15 Iteration: 15

 Hello from thread = 0 Iteration: 0

 Hello from thread = 4 Iteration: 4

 Hello from thread = 8 Iteration: 8


 GoodBye - Team Destroyed - Exiting Program
```

With 5 threads:



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc fourth.c -fopenmp && ./a.out

 Hello from thread = 0 Iteration: 0

 Hello from thread = 0 Iteration: 1

 Hello from thread = 0 Iteration: 2

 Hello from thread = 0 Iteration: 3

 Hello from thread = 4 Iteration: 13

 Hello from thread = 4 Iteration: 14

 Hello from thread = 4 Iteration: 15

 Hello from thread = 2 Iteration: 7

 Hello from thread = 2 Iteration: 8

 Hello from thread = 2 Iteration: 9

 Hello from thread = 3 Iteration: 10

 Hello from thread = 1 Iteration: 4

 Hello from thread = 1 Iteration: 5

 Hello from thread = 1 Iteration: 6

 Hello from thread = 3 Iteration: 11

 Hello from thread = 3 Iteration: 12


 GoodBye - Team Destroyed - Exiting Program
```

The outputs above clearly show that for thread counts of 2, 4, 8, and 16, the iterations were distributed evenly. However, when the thread count was increased to 32, the additional threads (those with thread numbers 16 and above) did not receive any iterations and remained unused. When the number of threads was an odd number, the iterations were not distributed evenly. For instance, when the thread count was 5, the first thread (thread number 0) was assigned 4 iterations, while the remaining threads each received 3 iterations.

5. ***Percent of Incorrect Answers Generated by Flawed Parallel OpenMP Program***

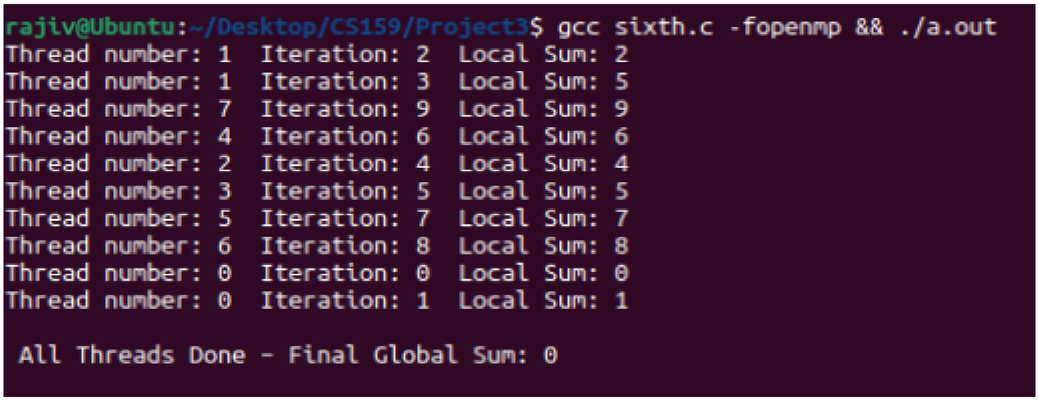
COUNT	NTHREADS=2	NTHREADS=4	NTHREADS=8	NTHREADS=32	NTHREADS=64
10	0%	0%	0%	0%	0%
100	0%	0%	0%	0%	10%
1000	0%	0%	0%	10%	30%

Certain results are inaccurate due to the "sum" variable being accessed by all threads simultaneously. There could be instances where one thread is modifying the value of this variable while another thread is in the process of reading it. This leads to unpredictable behavior and erroneous outcomes.

For lower counts (10 and 100), regardless of the number of threads, the program generates 0% incorrect answers. This suggests that the program performs well for lower counts.

However, as the count increases to 1000, we start to see some incorrect answers. This trend suggests that as both the count and the number of threads increase, the likelihood of the program generating incorrect answers also increases. This could be due to issues with thread synchronization or data races in the program.

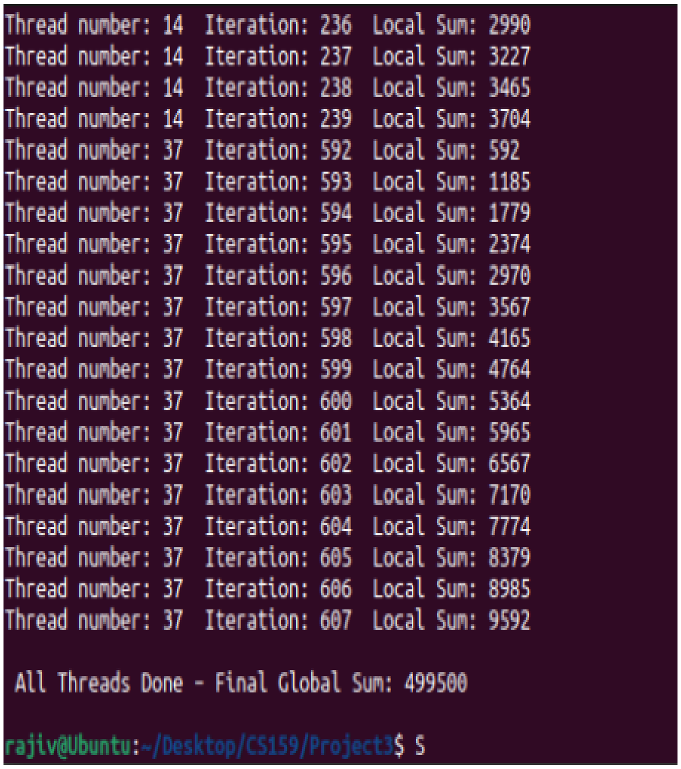
6. With 8 threads and 10 count:



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc sixth.c -fopenmp && ./a.out
Thread number: 1  Iteration: 2  Local Sum: 2
Thread number: 1  Iteration: 3  Local Sum: 5
Thread number: 7  Iteration: 9  Local Sum: 9
Thread number: 4  Iteration: 6  Local Sum: 6
Thread number: 2  Iteration: 4  Local Sum: 4
Thread number: 3  Iteration: 5  Local Sum: 5
Thread number: 5  Iteration: 7  Local Sum: 7
Thread number: 6  Iteration: 8  Local Sum: 8
Thread number: 0  Iteration: 0  Local Sum: 0
Thread number: 0  Iteration: 1  Local Sum: 1

 All Threads Done - Final Global Sum: 0
```

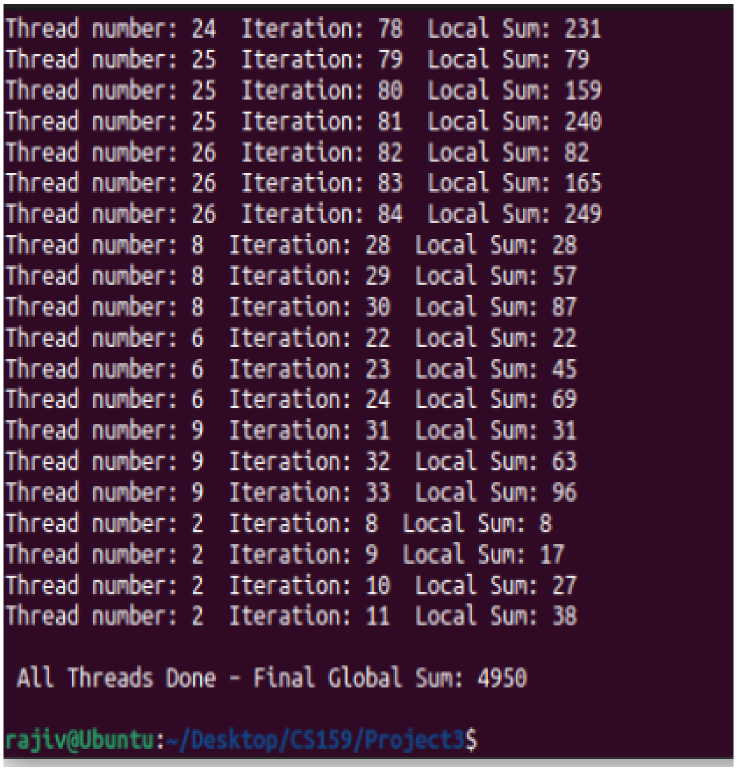
When the “sum” variable is made private, it means that each thread has its own separate copy of the variable. As a result, each thread is just adding to an initial value of 0, without interfering with the “sum” variable of other threads. In the end, the parent thread displays the value of sum, which remains 0. So, even though the outcome is predictable, it doesn’t accomplish the intended goal.

7. With 64 threads and 1000 count:



```
Thread number: 14  Iteration: 236  Local Sum: 2990
Thread number: 14  Iteration: 237  Local Sum: 3227
Thread number: 14  Iteration: 238  Local Sum: 3465
Thread number: 14  Iteration: 239  Local Sum: 3704
Thread number: 37  Iteration: 592  Local Sum: 592
Thread number: 37  Iteration: 593  Local Sum: 1185
Thread number: 37  Iteration: 594  Local Sum: 1779
Thread number: 37  Iteration: 595  Local Sum: 2374
Thread number: 37  Iteration: 596  Local Sum: 2970
Thread number: 37  Iteration: 597  Local Sum: 3567
Thread number: 37  Iteration: 598  Local Sum: 4165
Thread number: 37  Iteration: 599  Local Sum: 4764
Thread number: 37  Iteration: 600  Local Sum: 5364
Thread number: 37  Iteration: 601  Local Sum: 5965
Thread number: 37  Iteration: 602  Local Sum: 6567
Thread number: 37  Iteration: 603  Local Sum: 7170
Thread number: 37  Iteration: 604  Local Sum: 7774
Thread number: 37  Iteration: 605  Local Sum: 8379
Thread number: 37  Iteration: 606  Local Sum: 8985
Thread number: 37  Iteration: 607  Local Sum: 9592

 All Threads Done - Final Global Sum: 499500

rajiv@Ubuntu:~/Desktop/CS159/Project3$ S
```

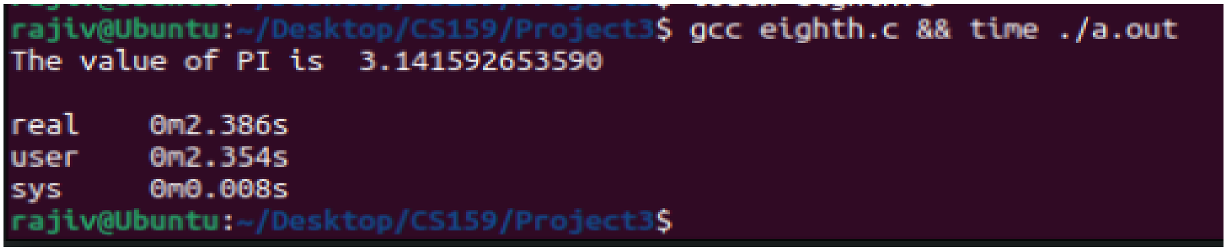
With 32 threads and 100 count



```
Thread number: 24  Iteration: 78  Local Sum: 231
Thread number: 25  Iteration: 79  Local Sum: 79
Thread number: 25  Iteration: 80  Local Sum: 159
Thread number: 25  Iteration: 81  Local Sum: 240
Thread number: 26  Iteration: 82  Local Sum: 82
Thread number: 26  Iteration: 83  Local Sum: 165
Thread number: 26  Iteration: 84  Local Sum: 249
Thread number: 8  Iteration: 28  Local Sum: 28
Thread number: 8  Iteration: 29  Local Sum: 57
Thread number: 8  Iteration: 30  Local Sum: 87
Thread number: 6  Iteration: 22  Local Sum: 22
Thread number: 6  Iteration: 23  Local Sum: 45
Thread number: 6  Iteration: 24  Local Sum: 69
Thread number: 9  Iteration: 31  Local Sum: 31
Thread number: 9  Iteration: 32  Local Sum: 63
Thread number: 9  Iteration: 33  Local Sum: 96
Thread number: 2  Iteration: 8  Local Sum: 8
Thread number: 2  Iteration: 9  Local Sum: 17
Thread number: 2  Iteration: 10  Local Sum: 27
Thread number: 2  Iteration: 11  Local Sum: 38

 All Threads Done - Final Global Sum: 4950

rajiv@Ubuntu:~/Desktop/CS159/Project3$
```

As shown in the above outputs, the issue of non-determinism is resolved by using reduction(+:sum). This operation guarantees that each parallel thread maintains its own local copy of the “sum” variable. Once all threads have completed their execution, these local copies are combined to yield the final value of “sum”.

8. Program took about 2.3 seconds to complete.



```
rajiv@Ubuntu:~/Desktop/CS159/Project3$ gcc eighth.c && time ./a.out
The value of PI is  3.141592653590

real    0m2.386s
user    0m2.354s
sys     0m0.008s
rajiv@Ubuntu:~/Desktop/CS159/Project3$
```

9. Below table shows time taken for different values of NTHREADS

NTHREADS	Time Taken (in seconds)
2	1.478
4	1.176
8	1.161
16	1.186
32	1.193
64	1.203
128	1.142

All the above tests were taken in a 2-core machine. (machine details in Page 1.)

The above tests indicate that the computation time for calculating PI significantly reduces(from sequential to parallel) until the thread count matches the number of cores, which is 2 threads in this scenario. However, beyond this point, increasing the number of threads doesn’t result in substantial changes in execution time.