**Lab-Report for Lab-3 (submitted by Parth Parashar)**

This is a great lab for learning the workings of OPENMP.

I have never worked with OPENMP and this is a great learning curve for me. The first few exercises tested my basic knowledge of OPENMP.

I did not have any particular trouble with Hello-World (hello-omp.c). It was a great starting point to start working with OpenMP.

The loop-nest did give me some troubles because of the way the threads were working. It became very vague at first and I had to go step by step to understand what was going on. But once I understood the flow of it, It became a little easier. It helped me understand the critical directive.

This helped me in the next questions as well. The recursive routines did give me some troubles as recursion in itself is a tricky concept and when paired with parallelism, it became even trickier. But with the help of sections, it became a little easier to implement.

The lock-ownership program helped me immensely to understand the concept of lock in a practical parallel environment.

The next set of questions were a little tricky. It helped me gain invaluable experience of working on the sections directive, task directive, for directive and how these concepts function to perform task and nested parallelism.

I did get stuck on the sections-parallelism because the sections have to be defined within the scope of the “sections” directive which I was not doing. After spending quite a bit of time on it, I was able to determine the importance of scope in sections directive.

The bank-omp.c and prime-omp.cpp programs were terrific in stimulating our brain to apply the OPENMP directives in some real-world scenario problems.

I had to spend a lot of time on it to understand how to approach such problems and which directives can be used where to achieve the desired results.

Overall, I had fun learning about sections, for directives, tasks and how OPENMP is useful in implementing parallel programming problems.

**Hello World (hello-omp.c)**

1. Upon analysing the code, we can safely say that 8 copies of Hello World!! will be printed. This is because it is running on a server with 8 cores and since we have not provided any argument for the number of threads in the directive, it will be equal to the number of cores.

The code along with the output is given below: -

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

1. A) Environment Variable

This can be done by using the export statement and then running the program as it is.

export OMP\_NUM\_THREADS=7

./hello-omp

The screenshot below gives the result of these commands

Text

Description automatically generated

B) In-Program Directive

This can be done by adding the in-program directive: -

#pragma omp parallel num\_threads(4)

The modified code is given below: -

Text

Description automatically generated

Upon compiling and running the program, we get the following output: -

Text

Description automatically generated

Scenario: - When both the environment variables and the in-program directives are changed and run, then, the preference is given to the in-program directive.

1. The code for getting the required output is given below: -

Text

Description automatically generated

When this is compiled and run, we get the following output: -

Graphical user interface, text

Description automatically generated

**LOOP NEST (loop-omp\*.c)**

The code for loop.c is given below: -

Text

Description automatically generated

The output of this program after compiling and running it is given below: -

Text

Description automatically generated

1. The code for loop-omp1.c is given below: -

Text

Description automatically generated

The output of this program after compiling and running is given below: -

Run-1

Text

Description automatically generated

Run-2

Text

Description automatically generated

As we can see that with every run the result is different. This is because of the variable total which is being simultaneously changed by the threads and hence the wrong output.

The code for loop-omp2.c is given below: -

Text

Description automatically generated

The output for this code after compilation is given below: -

Run-1: -

A picture containing text

Description automatically generated

Run-2: -

Text

Description automatically generated

We can see that the outputs vary with each run

1. The code for loop-omp3.c is: -
2. After modifying the code for loop-omp1.c by adding the critical directive to the variable total, we get,

Text

Description automatically generated

The output of this code is: -

Text

Description automatically generated with medium confidence

1. After adding the critical directive to the total variable for loop-omp2, we get,

Text

Description automatically generated

The output of this is given below: -

Text

Description automatically generated

**RECURSIVE ROUTINES(rec-omp\*.c)**

Code and output for rec.c is given below: -

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

1. rec-omp1.c

The code for rec-omp1.c is given below: -

Text

Description automatically generated

Text

Description automatically generated

As we can see from the output that output is in different order each time the program is run.

Rec-omp2.c

The code and output after running the code is given below: -

Text

Description automatically generated

No, I do not see the expected parallelism in both codes as only thread executes the task and then prints it.

1. A) Modifications to the rec-omp1.c to support parallel programming

Text

Description automatically generated

The output when run multiple times is given below: -

Graphical user interface, text

Description automatically generated

Modifications to the rec-omp2.c to achieve parallelism is given below: -

Text

Description automatically generated

The output is given below: -

Text

Description automatically generated

**LOCK OWNERSHIP(lock-omp.c)**

The code given in the lock-omp.c file is given below: -

Text

Description automatically generated

To check whether the lock locked by thread A is unlocked by thread B, we can compile and run the program and if it goes into an infinite loop, then the condition is validated.

Upon compiling and running the program, we have the following output: -

A picture containing table

Description automatically generated

Here, we can see that the lock is now reset by the other thread.

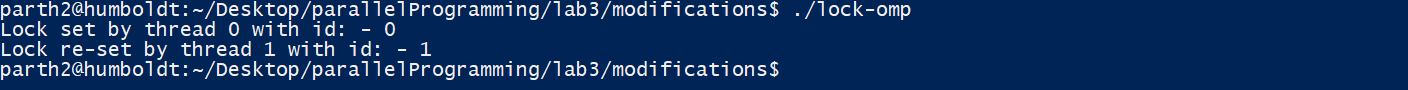
This can also be verified by checking the thread number which invokes the lock.

The code for the same is given below: -

Text

Description automatically generated

The output when this code is run is: -



**NESTED PARALLELISM (nested-omp.c)**

1)

The code is given as: -

Text

Description automatically generated

This code produces 4 lines of output as given below: -

Graphical user interface, text, application, email

Description automatically generated

The output has randomly selected threads running the function. This is because when threads are spawned, they all have the access to the same critical section and same memory area. This means each one of them will execute the resource as and when they get it.

The two parallel directives were run and they were allocated the critical section which leads to a race between the threads and whichsoever is invoked at the end gets the resource and runs it.

2) When the variable OMP\_NESTED is turned to true, then the following output is produced.

Text

Description automatically generated

This is because when the OMP\_NESTED is turned to true, the threads are run in parallel and each has its own mechanism to prevent the overriding of obtaining the critical section.

**TASK PARALLELISM (bank-omp.c)**

1. A) sections:-

The code for the same along with the output is given below in the screenshots

Text

Description automatically generated

Text

Description automatically generated

The output is as expected and is given below: -

Text

Description automatically generated

B) Tasks

The code and the output is given below: -

Text

Description automatically generated

Text

Description automatically generated

The output is given below: -

Text

Description automatically generated

1. For loop

The code and the output is given below: -

Text

Description automatically generated

Text

Description automatically generated

1. There is a need for synchronization because the threads use a lot of directives and that makes the task of debugging difficult. Also, the process slows down as well.

To implement synchronization, I would use task and taskwithdraw.

The code and the output is given below: -

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**PRIME FINDING (prime.cpp)**

The code given is: -

Text

Description automatically generated

The output for the same is given below: -A picture containing graphical user interface

Description automatically generated