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

High Performance Computing Lab

Class: Final Year (Computer Science and Engineering) **Year:** 2022-23

PRN: 2019BTECS00089 – Piyush Pramod Mhaske **Batch**: B3

Practical No. 4

github link: https://github.com/Piyush4620/2019BTECS00089HPCLab

To analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

Problem Statement 1:

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

int fib(int n)

{
  int f[n+2];
  int i;
  #pragma omp task
  {
    f[0] = 0;
    f[1] = 1;
  }
  #pragma omp task
  {
    for (i = 2; i <= n; i++)
  {
    f[i] = f[i-1] + f[i-2];
  }
  }
  #pragma omp taskwait
  {
  return f[n];
}
}


int main ()

{
```

```
  int n = 9;
  #pragma omp parallel
  {
  printf("%d", fib(n));
  }
  getchar();
  return 0;
}
```

Output:

```
PS D:\Academics\Fourth Year\HPC Lab\Assignments>  & 'c:\Users\PIYUSH\.vscode\extensions\ms-vscode.cpptools-1.12.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-44ligo4k.4uo' '--stdout=Microsoft-MIEngine-Out-1cixy2t3.gyf' '--stderr=Microsoft-MIEngine-Error-cykxk3vr.ebq' '--pid=Microsoft-MIEngine-Pid-yz5g2qkp.equ' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'
55
```

Information 1:

The shared clause declares the variables in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variables.The firstprivate clause provides a superset of the functionality provided by the private clause. The private variable is initialized by the original value of the variable when the parallel construct is encountered.

Problem Statement 2:

Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements.

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

int mutex = 1;
```

```
int full = 0;

int empty = 10, x = 0;

void producer()
{
  --mutex;

  ++full;

  --empty;

  x++;
  printf("\nProducer produces"
    "item %d",
    x);

  ++mutex;
}

void consumer()
{
  --mutex;

  --full;

  ++empty;
  printf("\nConsumer consumes "
    "item %d",
    x);
  x--;

  ++mutex;
}

int main()
{
  int n, i;
  printf("\n1. Press 1 for Producer"
    "\n2. Press 2 for Consumer"
    "\n3. Press 3 for Exit");

#pragma omp critical

  for (i = 1; i > 0; i++) {

    printf("\nEnter your choice:");
    scanf("%d", &n);

    switch (n) {
    case 1:
      if ((mutex == 1)
        && (empty != 0)) {
        producer();
```

```
      }

      else {
        printf("Buffer is full!");
      }
      break;

    case 2:
      if ((mutex == 1)
        && (full != 0)) {
        consumer();
      }
      else {
        printf("Buffer is empty!");
      }
      break;

    case 3:
      exit(0);
      break;
    }
  }
}
```

Output:

```
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical4> gcc -o producerConsumer -fopenmp producerConsumer.c
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical4> .\producerConsumer.exe

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:1

Producer producesitem 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:3
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical4>
```

Information 2:

A thread waits at the start of a critical region identified by a given name until no other thread in the program is executing a critical region with that same name. Critical sections not specifically named by omp critical directive invocation are mapped to the same unspecified name.

github link:
https://github.com/Piyush4620/2019BTECS00089HPCLab