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

Class: *Final Year (Computer Science and Engineering)*

Year: *2024-25* **Semester:** *1*

Course: *High Performance Computing Lab*

Practical No. 4

Exam Seat No:21510042

Omkar Rajesh Auti

CSE

Title of practical:

Study and Implementation of Synchronization

Problem Statement 1:

Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

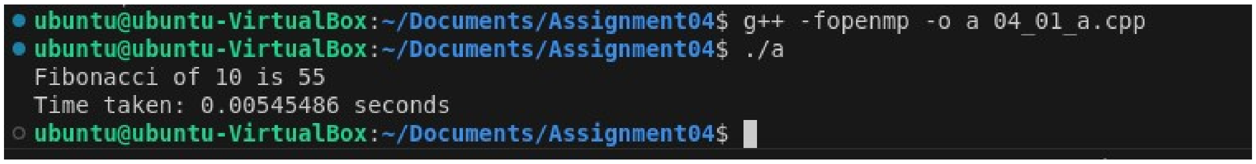
Fibonacci Computation:

Screenshots:

Code:

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

// Function to compute Fibonacci number using recursion
int fibonacci(int n) {
if (n <= 1)
return n;
int x, y;
// Parallelize the recursive calls
#pragma omp parallel sections
{
#pragma omp section
x = fibonacci(n - 1);
#pragma omp section
y = fibonacci(n - 2);
}

return x + y;
}

int main() {
int n = 10;// Example input
// Set the number of threads
omp_set_num_threads(2);
double start_time = omp_get_wtime(); // Start time measurement
int result = fibonacci(n);
double end_time = omp_get_wtime(); // End time measurement
std::cout << "Fibonacci of " << n << " is " << result << std::endl;
std::cout << "Time taken: " << (end_time - start_time) << " seconds" << std::endl;

return 0;
}
```

Output:



```
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$ g++ -fopenmp -o a 04_01_a.cpp
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$ ./a
Fibonacci of 10 is 55
Time taken: 0.00545486 seconds
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$
```

Information:

While parallelizing small problems might not always yield performance gains due to overhead, the approach can be very effective for larger, more computationally intensive problems.

Problem Statement 2:

Analyse and implement a Parallel code for below programs using OpenMP considering synchronization requirements. (Demonstrate the use of different clauses and constructs wherever applicable)

Producer Consumer Problem

Screenshots:

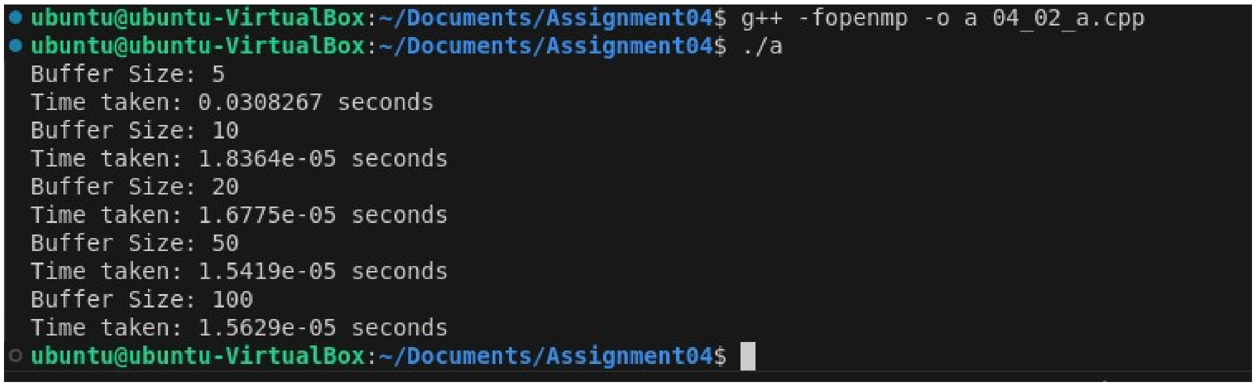
Code:

```
#include <iostream>
#include <omp.h>
#include <queue>
#include <cstdlib>
#include <ctime>

const int NUM_PRODUCERS = 2;
const int NUM_CONSUMERS = 2;
const int NUM_ITEMS = 100;

std::queue<int> buffer;
omp_lock_t buffer_lock;
bool buffer_not_full = true;
bool buffer_not_empty = false;
int count = 0;

void producer(int id, int buffer_size) {
for (int i = 0; i < NUM_ITEMS / NUM_PRODUCERS; ++i) {
int item = rand() % 100;
omp_set_lock(&buffer_lock);
// Wait if buffer is full
while (buffer.size() == buffer_size) {
buffer_not_full = false;
omp_unset_lock(&buffer_lock);
```

```
while (!buffer_not_full) { /* Busy wait */ }
omp_set_lock(&buffer_lock);
}
buffer.push(item);
count++;
// std::cout << "Producer " << id << " produced " << item << std::endl;
buffer_not_empty = true;

omp_unset_lock(&buffer_lock);
}
}

void consumer(int id, int buffer_size) {
for (int i = 0; i < NUM_ITEMS / NUM_CONSUMERS; ++i) {
int item;
omp_set_lock(&buffer_lock);
// Wait if buffer is empty
while (buffer.empty()) {
buffer_not_empty = false;
omp_unset_lock(&buffer_lock);
while (!buffer_not_empty) { /* Busy wait */ }
omp_set_lock(&buffer_lock);
}
item = buffer.front();
buffer.pop();
count--;
// std::cout << "Consumer " << id << " consumed " << item << std::endl;
buffer_not_full = true;

omp_unset_lock(&buffer_lock);
}
}

void run_test(int buffer_size) {
buffer = std::queue<int>(); // Reset buffer
count = 0; // Reset item count
buffer_not_full = true;
buffer_not_empty = false;

omp_init_lock(&buffer_lock);
double start_time = omp_get_wtime();
#pragma omp parallel
{
#pragma omp sections
{
```

```
#pragma omp section
{
for (int i = 0; i < NUM_PRODUCERS; ++i) {
#pragma omp parallel
producer(i, buffer_size);
}
}
#pragma omp section
{
for (int i = 0; i < NUM_CONSUMERS; ++i) {
#pragma omp parallel
consumer(i, buffer_size);
}
}
}
}
double end_time = omp_get_wtime();
omp_destroy_lock(&buffer_lock);
std::cout << "Buffer Size: " << buffer_size << std::endl;
std::cout << "Time taken: " << (end_time - start_time) << " seconds" << std::endl;
}

int main() {
srand(time(0));
// Test different buffer sizes
int buffer_sizes[] = {5, 10, 20, 50, 100};
int num_tests = sizeof(buffer_sizes) / sizeof(buffer_sizes[0]);
for (int i = 0; i < num_tests; ++i) {
int buffer_size = buffer_sizes[i];
run_test(buffer_size);
}
return 0;
}
```

Output:



```
● ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$ g++ -fopenmp -o a 04_02_a.cpp
● ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$ ./a
  Buffer Size: 5
  Time taken: 0.0308267 seconds
  Buffer Size: 10
  Time taken: 1.8364e-05 seconds
  Buffer Size: 20
  Time taken: 1.6775e-05 seconds
  Buffer Size: 50
  Time taken: 1.5419e-05 seconds
  Buffer Size: 100
  Time taken: 1.5629e-05 seconds
○ ubuntu@ubuntu-VirtualBox:~/Documents/Assignment04$ 
```

Information:

For larger size of buffer --> Parallel programs time not incresing significantly

Github Link:

https://github.com/omkarauti11/HPC_LAB