CS3811 - High Performance Computing and Big Data Lab

Lab 5

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

Write a C++ program to find the nth factorial and parallelize it using OpenMP.

Code

Written in C++.

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

unsigned long long factorial_serial(int n) {
    if (n <= 1) {
        return n;
    }

    return n * factorial_serial(n - 1);
}

unsigned long long factorial_parallel(int n) {
    if (n <= 1) {
        return n;
    }

    unsigned long long res = 1;
    #pragma omp parallel shared(res)
    {
        #pragma omp single nowait
        {
            #pragma omp parallel shared(res)
            {
                res = n * factorial_parallel(n - 1);
            }
```

```cpp
            }
        }

        #pragma omp taskwait
        return res;

        return n * factorial_serial(n - 1);
    }

    int main() {
        std::ofstream outfile("data5_0.dat");

        int n;
        std::cout << "Enter a positive number:" << std::endl;
        std::cin >> n;

        double start = omp_get_wtime();
        unsigned long long fact = factorial_serial(n);
        double duration = omp_get_wtime() - start;

        outfile << "Serial Code - " << duration << std::endl;

        start = omp_get_wtime();
        fact = factorial_parallel(n);
        duration = omp_get_wtime() - start;

        outfile << "Parallelized Code - " << duration << std::endl;

        outfile.close();

        return 0;
    }
```

Output

```
Enter a positive number:
10000
```

```
Serial Code - 0.000776028
Parallelized Code - 0.0352906
```

Experiment 2

Objective

Write a C++ program to find the nth number in the Fibonacci series.

Code

Writen in C++.

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

long long fib_serial(int n) {
    if (n <= 1) {
        return n;
    }

    return fib_serial(n - 1) + fib_serial(n - 2);
}

long long fib_parallel(int n) {
    if (n <= 1) {
        return n;
    }

    long long x, y;
    #pragma omp task shared(x)
    x = fib_parallel(n - 1);

    #pragma omp task shared(y)
    y = fib_parallel(n - 2);

    #pragma omp taskwait
    return x + y;
}

int main() {
    std::ofstream outfile("data5_1.dat");

    int n;
    std::cout << "Enter the position of the Fibonacci number to be printed:
" << std::endl;
    std::cin >> n;

    double start = omp_get_wtime();
    long long fib = fib_serial(n);
    double duration = omp_get_wtime() - start;

    outfile << "Serial Code - " << duration << std::endl;

    start = omp_get_wtime();
    fib = fib_parallel(n);
    duration = omp_get_wtime() - start;

    outfile << "Parallelized Code - " << duration << std::endl;

    outfile.close();
}
```

Output

```
Enter the position of the Fibonacci number to be printed:
30
```

```
Serial Code - 0.00926848
Parallelized Code - 0.0362814
```

Experiment 3

Objective

Write a C++ program to find matrix-matrix multiplication - parallelize using OpenMP. Employ different data distribution techniques.

Code

Writen in C++.

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

#define N 512

void init_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = 1;
        }
    }
}

void multiply_matrix(double A[N][N], double B[N][N], double C[N][N], int
op) {
    switch(op) {
        // serial
        case 0:
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                C[i][j] = 0;
                for (int k = 0; k < N; k++) {
                    C[i][j] += A[i][k]* B[k][j];
                }
            }
        }
        break;

        // static
        case 1:
        #pragma omp parallel for schedule(static)
```

```
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                C[i][j] = 0;
                for (int k = 0; k < N; k++) {
                    C[i][j] += A[i][k]* B[k][j];
                }
            }
        }
        break;

        //dynamic
        case 2:
        #pragma omp parallel for schedule(dynamic)
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                C[i][j] = 0;
                for (int k = 0; k < N; k++) {
                    C[i][j] += A[i][k]* B[k][j];
                }
            }
        }
        break;

        //guided
        case 3:
        #pragma omp parallel for schedule(guided)
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                C[i][j] = 0;
                for (int k = 0; k < N; k++) {
                    C[i][j] += A[i][k]* B[k][j];
                }
            }
        }
        break;

        default:
        std::cout << "Invalid option!" << std::endl;
        break;
    }
}

int main() {
    std::ofstream outfile("data5_2.dat");


    double A[N][N], B[N][N], C[N][N];

    init_matrix(A);
    init_matrix(B);

    // serial
    double start = omp_get_wtime();
    multiply_matrix(A, B, C, 0);
```

```cpp
    double duration = omp_get_wtime() - start;
    outfile << "Serial Code - " << duration << std::endl;


    // static
    start = omp_get_wtime();
    multiply_matrix(A, B, C, 1);
    duration = omp_get_wtime() - start;
    outfile << "Parallel Static Code - " << duration << std::endl;

    // dynamic
    start = omp_get_wtime();
    multiply_matrix(A, B, C, 2);
    duration = omp_get_wtime() - start;
    outfile << "Parallel Dynamic Code - " << duration << std::endl;

    // guided
    start = omp_get_wtime();
    multiply_matrix(A, B, C, 3);
    duration = omp_get_wtime() - start;
    outfile << "Parallel Guided Code - " << duration << std::endl;

    outfile.close();

    return 0;
}
```

Output

```
Serial Code - 0.502835
Parallel Static Code - 0.150492
Parallel Dynamic Code - 0.151288
Parallel Guided Code - 0.132702
```