

Class: Final Year (Computer Science and Engineering)

Year: 2023-24

Semester: 1

Course: High Performance Computing Lab

Practical No. 2

Exam Seat No: 2020BTECS00006

Name: Samrat V Jadhav

Title of practical: Study and implementation of basic OpenMP clauses

Implement following Programs using OpenMP with C:

1. Vector Scalar Addition
2. Calculation of value of Pi

Analyse the performance of your programs for different number of threads and Data size.

Problem Statement 1: Vector Scalar Addition

Screenshots:

Vector Scalar Addition Sequential Code:

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

void main()
{
    int N = 100;
    int A[N];
    for(int i=0;i<N;i++)A[i] = i + 1;
    int S = 2000;

    double itime, ftime, exec_time;
    itime = omp_get_wtime();
    for (int i = 0; i < N; i++)
    {
        A[i] += S;
        printf("Thread: %d Index: %d\n", omp_get_thread_num(),i);
    }
}
```

```
ftime = omp_get_wtime();
exec_time = ftime - itime;

// for(int i=0;i<N;i++){
//     printf("%d ", A[i]);
// }

printf("\nTime taken is %f\n", exec_time);
printf("\n");
}
```

Vector Scalar Addition Sequential Output:

```
PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp vectscaladdseq.c
PS D:\Final BTech Labs\Assignment 2> .\a
Thread: 0 Index: 0
Thread: 0 Index: 1
Thread: 0 Index: 2
Thread: 0 Index: 3
Thread: 0 Index: 4
Thread: 0 Index: 5
Thread: 0 Index: 6
Thread: 0 Index: 7
Thread: 0 Index: 8
Thread: 0 Index: 9
Thread: 0 Index: 10
Thread: 0 Index: 11
Thread: 0 Index: 12
Thread: 0 Index: 13
Thread: 0 Index: 14
Thread: 0 Index: 15
Thread: 0 Index: 16
Thread: 0 Index: 17
Thread: 0 Index: 18
Thread: 0 Index: 19
Thread: 0 Index: 20
Thread: 0 Index: 93
Thread: 0 Index: 94
Thread: 0 Index: 95
Thread: 0 Index: 96
Thread: 0 Index: 97
Thread: 0 Index: 98
Thread: 0 Index: 99

Time taken is 0.041000
```

Vector Scalar Addition Parallel Code:

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

void main()
{
```

```
int N = 100;
int A[N];
for(int i=0;i<N;i++)A[i] = i + 1;
int S = 212354454;

omp_set_num_threads(6);

double itime, ftime, exec_time;
itime = omp_get_wtime();
#pragma omp parallel for
for (int i = 0; i < N; i++)
{
    A[i] += S;
    // printf("Thread: %d Index: %d\n", omp_get_thread_num(),i);
}
ftime = omp_get_wtime();
exec_time = ftime - itime;

// for(int i=0;i<N;i++){
//     printf("%d ", A[i]);
// }


printf("\nTime taken is %f\n", exec_time);
}
```

Vector Scalar Addition Parallel Output:

```
PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp vectscaladdpar.c
PS D:\Final BTech Labs\Assignment 2> .\a
Thread: 0 Index: 0
Thread: 0 Index: 1
Thread: 0 Index: 2
Thread: 0 Index: 3
Thread: 2 Index: 41
Thread: 2 Index: 42
Thread: 2 Index: 43
Thread: 2 Index: 44
Thread: 2 Index: 45
Thread: 2 Index: 46
Thread: 2 Index: 47
Thread: 2 Index: 48
Thread: 2 Index: 49
Thread: 2 Index: 50
Time taken is 0.040000
PS D:\Final BTech Labs\Assignment 2>
```


Information:

Execution time for sequential processing is:



Time taken is 0.041000

Execution time for parallel processing is:



Time taken is 0.001000

Analysis:

Increasing the thread count beyond the number of CPU cores can potentially reduce execution time up to a point. Beyond that point, excessive threads may introduce overhead. Changing the thread count won't directly affect execution time since it's fixed at 6 threads. However, execution time can still vary depending on hardware and workload characteristics.

Problem Statement 2: Calculation of value of Pi

Screenshots:

Calculation of value of Pi Sequential Code:

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

#define NUM_POINTS 1000000

int main() {
    srand(time(NULL));

    clock_t start_time = clock();

    int inside_circle = 0;

    for (int i = 0; i < NUM_POINTS; i++) {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;
        double distance = x * x + y * y;
```

```
        if (distance <= 1.0) {
            inside_circle++;
        }
    }

    double pi = 4.0 * inside_circle / NUM_POINTS;
    printf("Estimated Pi value (sequential): %lf\n", pi);

    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Execution time (sequential): %lf seconds\n", execution_time);

    return 0;
}
```

Calculation of value of Pi Sequential Output:

```
● PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp piseq.c
● PS D:\Final BTech Labs\Assignment 2> .\a
Estimated Pi value (sequential): 3.144136
Execution time (sequential): 0.069000 seconds
```

Calculation of value of Pi Parallel Code:

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

#define NUM_POINTS 1000000

int main() {
    srand(time(NULL));

    clock_t start_time = clock();

    int inside_circle = 0;

    #pragma omp parallel for reduction(+:inside_circle)
    for (int i = 0; i < NUM_POINTS; i++) {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;
        double distance = x * x + y * y;
    }
}
```

```
        if (distance <= 1.0) {
            inside_circle++;
        }
    }

    double pi = 4.0 * inside_circle / NUM_POINTS;
    printf("Estimated Pi value (parallel): %lf\n", pi);

    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
    printf("Execution time (parallel): %lf seconds\n", execution_time);

    return 0;
}
```

Calculation of value of Pi Parallel Output:

```
PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp pipar.c
● PS D:\Final BTech Labs\Assignment 2> .\a
Estimated Pi value (parallel): 3.139948
Execution time (parallel): 0.028000 seconds
```

Information:

Execution time for sequential processing is:

```
● PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp piseq.c
● PS D:\Final BTech Labs\Assignment 2> .\a
Estimated Pi value (sequential): 3.144136
Execution time (sequential): 0.069000 seconds
```

Execution time for parallel processing is:

```
PS D:\Final BTech Labs\Assignment 2> gcc -fopenmp pipar.c
● PS D:\Final BTech Labs\Assignment 2> .\a
Estimated Pi value (parallel): 3.139948
Execution time (parallel): 0.028000 seconds
```

Analysis:

Increasing the thread count beyond the number of CPU cores can potentially reduce execution time up to a point. Beyond that point, excessive threads may introduce overhead. Changing the thread count won't directly affect execution time since it's fixed at 6 threads. However, execution time can still vary depending on hardware and workload characteristics.

Github Link: