

## CSE 5250 Assignment 1

1.) Pi approximation C file program:

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

long long int n = 1000000;
long long int thread_count;
double sum = 0;

int flag = 0;

void * Thread_sum(void* rank);

int main(int argc, char* argv[])
{
    long long int thread;
    pthread_t* thread_handles;

    thread_count = strtol(argv[1], NULL, 10);

    thread_handles = (pthread_t*)malloc(thread_count * sizeof(pthread_t));

    for (thread = 0; thread < thread_count; thread++)
    {
        pthread_create(&thread_handles[thread], NULL, Thread_sum, (void*)thread);
    }

    for (thread = 0; thread < thread_count; thread++)
    {
        pthread_join(thread_handles[thread], NULL);
    }

    double pi = 4.0 * sum;
    printf("Pi approximated using %llu terms is:\n", n);
    printf("%.18f\n", pi);

    free(thread_handles);
}
```

```

return 0;
}

void* Thread_sum(void* rank)
{
    long long int my_rank = (long long int)rank;
    double factor;
    long long int i;
    long long int my_n    = n / thread_count;
    long long int my_first_i = my_n * my_rank;
    long long int my_last_i  = my_first_i + my_n;

    if (my_first_i % 2 == 0) factor = 1.0;
    else factor = -1.0;

    for (i = my_first_i; i < my_last_i; i++, factor = -factor)
    {
        while (flag != my_rank);

        sum += factor / (2 * i + 1);

        flag = (flag + 1) % thread_count;
    }

    return NULL;
}

```

Image of the program running:

```

006896598@csusb.edu@jb358-7:~/CSE Parallel
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o HW1_Pi HW1_Pi.c
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./HW1_Pi 4
Pi approximated using 1000000 terms is:
3.141591653589772104
[006896598@csusb.edu@jb358-7 CSE Parallel]$

```

- 2.) When parallelization occurs, the loop-carried dependency and the variables declared before the loop causes them to be shared among each thread, causing erroneous numbers to be used in each thread calculation. To fix this problem, we must make each thread have its own copy of each variable by adding a private clause within the loop. A private clause will allow each thread to have its own copy of the variable.

3.) Image of programs running:

```
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o Fib Fib.c
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./Fib 4
The fibonacci numbers are:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, -200971056, -2009
71056, -401942112, -602913168, -1004855280, -1607768448, 1682343568, 74575120, 1756918688, 1831493808, -706554800, 112
4939008, 418384208, 1543323216, 1961707424, -789936656, 1171770768, 381834112, End of numbers reached.
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o Fib Fib.c
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./Fib 4
The fibonacci numbers are:
89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817,
39088169, 63245986, 102334155, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1, 1, 2, 3,
5, 8, 13, 21, 34, 55, End of numbers reached.
[006896598@csusb.edu@jb358-7 CSE Parallel]$
```

My predictions were incorrect, instead parallelizing the first for loop causes the program to print random numbers that are not a part of the Fibonacci sequence. Meanwhile, parallelizing the second for loop causes the program to print out the Fibonacci sequence in the wrong order.

#### 4.) Pi approximation different method C program:

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

int main(int argc, char* argv[])
{
    int thread_count = strtol(argv[1], NULL, 10);

    long long int num_in_circle;
    long long int num_of_tosses = 1000000;
    double pi_estimate;

    long long int toss;
    unsigned int seed;

    num_in_circle = 0;

    # pragma omp parallel num_threads(thread_count) private(seed)
    {
        int thread_id = omp_get_thread_num();
        seed = (unsigned int)(time(NULL) + thread_id);

        srand(seed);

    # pragma omp for
        for (toss = 0; toss < num_of_tosses; toss++)
        {
            float x = - 1 + 2 * (rand() / (double)RAND_MAX);
            float y = - 1 + 2 * (rand() / (double)RAND_MAX);

            float dist_squared = sqrt(x*x + y*y);

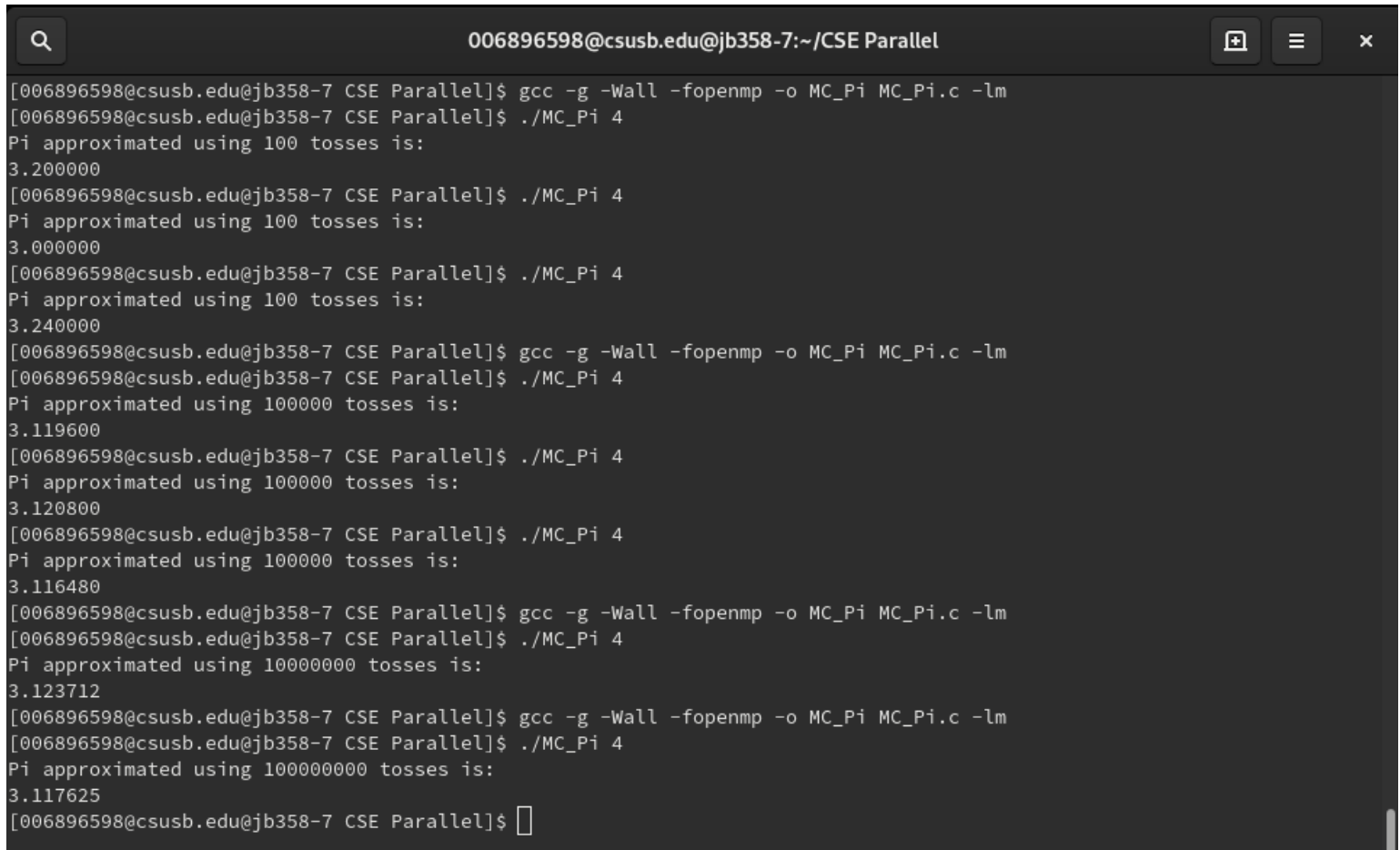
            if (dist_squared <= 1) num_in_circle++;
        }

        pi_estimate = 4 * num_in_circle / ((double) num_of_tosses);
    }

    printf("Pi approximated using %llu tosses is:\n", num_of_tosses);
    printf("%f\n", pi_estimate);
}
```

}

Image of the program running:



```
006896598@csusb.edu@jb358-7:~/CSE Parallel
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o MC_Pi MC_Pi.c -lm
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100 tosses is:
3.200000
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100 tosses is:
3.000000
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100 tosses is:
3.240000
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o MC_Pi MC_Pi.c -lm
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100000 tosses is:
3.119600
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100000 tosses is:
3.120800
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100000 tosses is:
3.116480
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o MC_Pi MC_Pi.c -lm
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 10000000 tosses is:
3.123712
[006896598@csusb.edu@jb358-7 CSE Parallel]$ gcc -g -Wall -fopenmp -o MC_Pi MC_Pi.c -lm
[006896598@csusb.edu@jb358-7 CSE Parallel]$ ./MC_Pi 4
Pi approximated using 100000000 tosses is:
3.117625
[006896598@csusb.edu@jb358-7 CSE Parallel]$
```