Problem2

Source code

omp_pi_one.c

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

long num_steps = 1000000000;
double step;

int main (void)
{
    long i; double x, pi, sum = 0.0;
    double start_time, end_time;

    omp_set_num_threads(1);
    start_time = omp_get_wtime();
    step = 1.0/(double) num_steps;
    for (i=0;i< num_steps; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
    end_time = omp_get_wtime();
    double timeDiff = end_time - start_time;
        printf("Execution Time : %.10lfsec\n", timeDiff);

    printf("pi=%.10lf\n",pi);
    return 0;
}
```

thrust_ex.cu

```c
#include <stdio.h>
#include <chrono>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

#include <thrust/copy.h>
#include <thrust/fill.h>

const long num_steps = 1000000000;

struct pi_fct
{
    const double step;
```

```
    pi_fct(double _step) : step(_step) {}

    __host__ __device__ double operator()(const long i) const
    {
        double x = (i + 0.5) * step;
        return 4.0 / (1.0 + x * x);
    }
};

int main(void)
{
    double pi = 0.0;
    double sum = 0.0;
    double step = 1.0 / (double)num_steps;
    double timeDiff;

    thrust::device_vector<int> numbers(num_steps);
    thrust::sequence(numbers.begin(), numbers.end());
    auto start = std::chrono::steady_clock::now();
    sum = thrust::transform_reduce(numbers.begin(), numbers.end(),
pi_fct(step), 0, thrust::plus<float>());
    pi = step * sum;
    auto end = std::chrono::steady_clock::now();
    timeDiff = std::chrono::duration<double>(end - start).count();
    printf("Execution Time : %.10lfsec\n", timeDiff);
    printf("pi=%.10lf\n", pi);
    return 0;
}
```

Execution time table

Execution type	Execution time (in sec)	Performance
1 thread	2.0944931060	0.47744248817
GPU	0.2491016600	4.01442527521

Interpretation

We can see that the GPU execution take less time that the one thread execution. So we can conclude that for this type of computation, GPU computation is better.