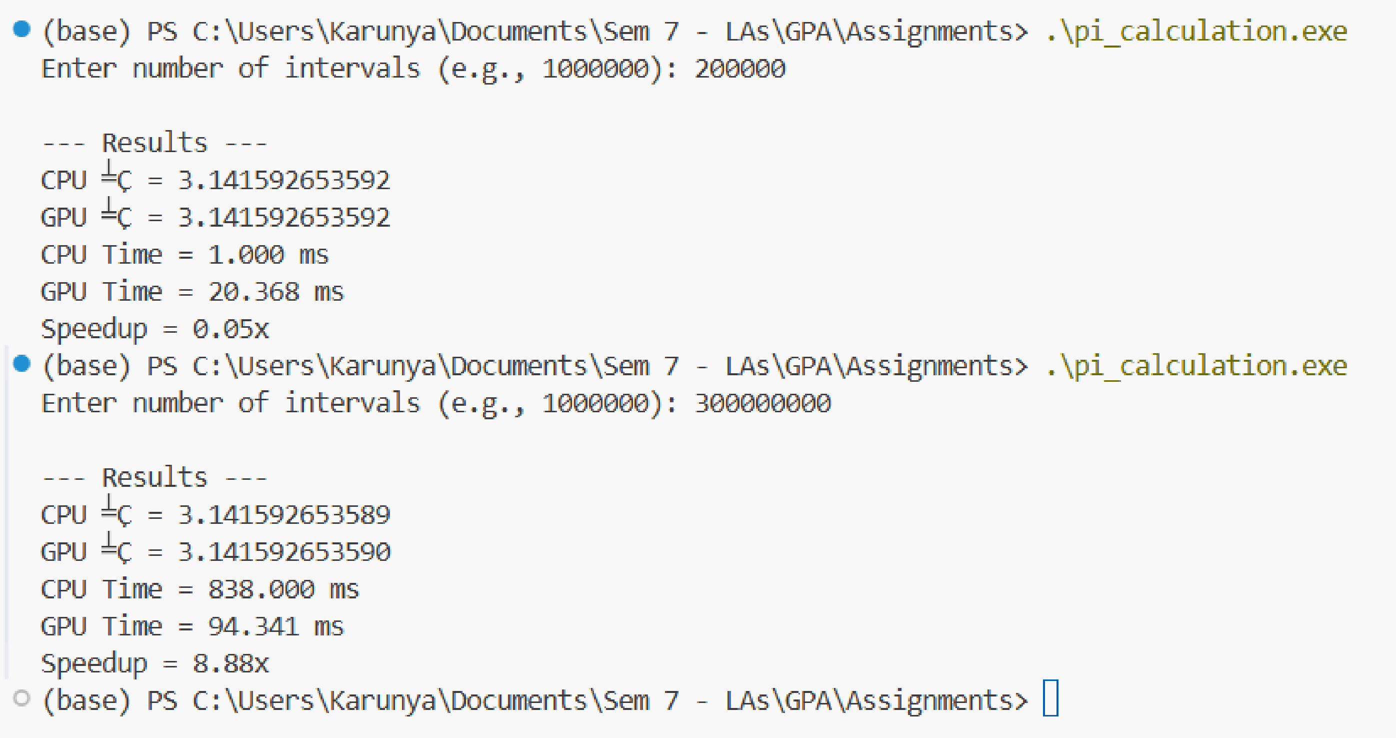
ASSIGNMENT - I

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

// GPU Kernel to compute partial sums for Pi
__global__ void computePiKernel(double step, double *sum, int n) {
    __shared__ double cache[256];
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int cacheIndex = threadIdx.x;

    double temp = 0.0;
    while (tid < n) {
        double x = (tid + 0.5) * step;
        temp += 4.0 / (1.0 + x * x);
        tid += blockDim.x * gridDim.x;
    }
    cache[cacheIndex] = temp;
    __syncthreads();

    // Parallel reduction in shared memory
    int i = blockDim.x / 2;
    while (i != 0) {
        if (cacheIndex < i)
            cache[cacheIndex] += cache[cacheIndex + i];
        __syncthreads();
        i /= 2;
    }

    if (cacheIndex == 0)
        sum[blockIdx.x] = cache[0];
```

```
}

int main() {
    int n;
    printf("Enter number of intervals (e.g., 1000000): ");
    scanf("%d", &n);

    double step = 1.0 / (double)n;

    // ================== CPU COMPUTATION ==================
    clock_t cpu_start = clock();
    double cpu_sum = 0.0;
    for (int i = 0; i < n; i++) {
        double x = (i + 0.5) * step;
        cpu_sum += 4.0 / (1.0 + x * x);
    }
    double pi_cpu = step * cpu_sum;
    clock_t cpu_end = clock();
    double cpu_time = ((double)(cpu_end - cpu_start)) / CLOCKS_PER_SEC * 1000.0; // ms

    // ================== GPU COMPUTATION ==================
    double *h_sum, *d_sum;
    h_sum = (double *)malloc(256 * sizeof(double));
    cudaMalloc((void **)&d_sum, 256 * sizeof(double));

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);
    computePiKernel<<<256, 256>>>(step, d_sum, n);
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    float gpu_time = 0.0f;
    cudaEventElapsedTime(&gpu_time, start, stop);

    cudaMemcpy(h_sum, d_sum, 256 * sizeof(double), cudaMemcpyDeviceToHost);

    double total = 0.0;
    for (int i = 0; i < 256; i++)
        total += h_sum[i];
    double pi_gpu = step * total;

    // ================= OUTPUT =================
    printf("\n--- Results ---\n");
    printf("CPU π = %.12f\n", pi_cpu);
    printf("GPU π = %.12f\n", pi_gpu);
    printf("CPU Time = %.3f ms\n", cpu_time);
    printf("GPU Time = %.3f ms\n", gpu_time);
    printf("Speedup = %.2fx\n", cpu_time / gpu_time);

    // Cleanup
    cudaFree(d_sum);
    free(h_sum);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    return 0;
}
```

OUTPUT



```
● (base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> .\pi_calculation.exe
  Enter number of intervals (e.g., 1000000): 200000

  --- Results ---
  CPU ╧Ç = 3.141592653592
  GPU ╧Ç = 3.141592653592
  CPU Time = 1.000 ms
  GPU Time = 20.368 ms
  Speedup = 0.05x
● (base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> .\pi_calculation.exe
  Enter number of intervals (e.g., 1000000): 300000000

  --- Results ---
  CPU ╧Ç = 3.141592653589
  GPU ╧Ç = 3.141592653590
  CPU Time = 838.000 ms
  GPU Time = 94.341 ms
  Speedup = 8.88x
○ (base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> 
```

Figure 1: Calculation of Pi Value at Different Interval.