Homework 1

Please finish the following problems using OpenMP. Note that your programs must be compilable and executable. Please provide a Makefile to compile all the program files each for a problem. You also need to write a README file on how to compile and run your programs.

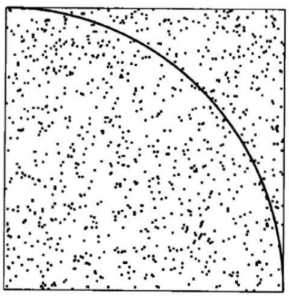
Problem 1

Floyd's algorithm is an $\theta(n^3)$ algorithm that solves the all-pairs shortest-path problem. Given an $n \times n$ ajacency matrix A, the following algorithm outputs the shortest path between every pair of vertices. Please use OpenMP to implement a parallel version of Floyd's algorithm.

```
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        A[i][j] = min(A[i][j], A[i][k] + A[k][j]);
      }
    }
}
```

Problem 2

We can use Monte Carlo method to compute π. The idea is to generate pairs of points in the unit square (where each coordinate varies between 0 and 1) .We count the fraction of points inside the circle(those points for which $x^2 + y^2 \leq 1$) .The expected value of this fraction is $\pi/4$;so multiplying the fraction by 4 gives an estimate of π.



Here is the C code implementing the algorithm:

```
    int count ;  //points inside the circle
    int i;
    int local_count;
    int samples;  //total points
    unsigned short xi[3];
    int t ; // number of threads
    int tid;
    double x,y;
```

```
    samples = atoi(argv[1]);
    xi[0] = atoi(argv[2]);
    xi[1] = atoi(argv[3]);
    xi[2] = atoi(argv[4]);
    count = 0;
    {
        for( i = tid; i < samples ;i+= t) {
            x = erand48(xi);
            y = erand48(xi);
            if(x*x + y*y <= 1.0 )
                count ++;
        }
    }

    printf("Estimate of pi : %7.5f \n",4.0 * count /samples);
```

Please use multiple threads to speed the execution of the above problem.

Hint: You must ensure that each thread is generating a different stream of random numbers . Otherwise ,each thread would generate the same sequence of (x,y) pairs,and there would be no increase in the precision of the answer through the use of parallelism.

Problem 3

In convolutional neural networks, the 2D convolution operation is a key computation. Given a 2D input matrix(size: MxN) and a convolution filter(size: KxK), the output matrix is obtained by performing a dot product between the filter and a sliding window over the input. The convolution operation can be parallelized to improve computational efficiency.

Use OpenMP to parallelize the convolution process. You are required to utilize reduction mechanisms to achieve this function. Use the "valid" mode to process boundaries (i.e., no padding, the output matrix size should be (M-K+1) x (N-K+1)).

Input:

```
// Example input matrix and convolution filter
float input[M][N];  // MxN input matrix
float filter[K][K]; // KxK convolution filter

// Output matrix
float output[M-K+1][N-K+1];  // Output matrix of size (M-K+1)x(N-K+1)
```

Hint: reduction is used to accumulate results during the dot product operation.