# 1 Hands-on; Shared Memory II; Synchronization Primitives

1. **A threaded program for computing the value of** $\pi$,

   - The method is used here is based on generating random numbers in a unit length square and counting the number of points that fall within the largest circle inscribed in the square.
   - Since the area of the circle ($\pi r^2$) is equal to $\pi/4$, and the area of the square is $1 \times 1$, the fraction of random points that fall in the circle should approach $\pi/4$.
   - A simple threaded strategy for generating the value of $\pi$ assigns a fixed number of points to each thread.
   - Each thread generates these random points and keeps track of the number of points that land in the circle locally.
   - After all threads finish execution, their counts are combined to compute the value of $\pi$ (by calculating the fraction over all threads and multiplying by 4).

   Vary the number of sample points and threads, then observe the outcome.

2. **A threaded program that determines the sum with the use of mutex variables**;

   - Increase the number of the threads and change the size of array.
   - Observe if the all the threads have the partial sum all the time. Why not?

3. **A threaded program that performs a dot product with the use of mutex variables**; ( sequential version)

   - First study the sequential version.
   - The main data is made available to all threads through a globally accessible structure.
   - Each thread works on a different part of the data.
   - The main thread waits for all the threads to complete their computations, and then it prints the resulting sum.

4. **This program demonstrates the use of condition variables**.

   - The main routine creates three threads.
   - Two of the threads perform work and update a “count” variable.
   - The third thread waits until the count variable reaches a specified value.