# Tarea 1: Calcular Pi usando pthreads

Esteban Rodríguez Betancourt

2024-03

## Introducción

El objetivo de esta tarea es calcular el valor de $\pi$ utilizando el método de Monte Carlo. Para ello, se generan puntos aleatorios en un cuadrado de lado 1 y se cuentan cuántos de estos puntos caen dentro de un círculo de radio 1. El valor de $\pi$ se puede aproximar mediante la siguiente fórmula:

$$\pi \approx 4 \times \frac{\text{puntos dentro del círculo}}{\text{puntos totales}}$$

Usen pthreads en C para paralelizar el cálculo de $\pi$. El programa debe recibir como argumento la cantidad de puntos a generar y el número de hilos a utilizar. El programa debe imprimir el valor de $\pi$ calculado y el tiempo que tomó realizar el cálculo.

## Implementación

Pueden usar el siguiente código como base para su implementación:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <math.h>
#include <stdint.h>

#define MAX_THREADS 50000

int total_threads;
int points_inside_circle = 0;
int total_points;

void* throw_darts(void* arg) {
    int childID = (uintptr_t)arg;
    int points_per_thread = total_points / total_threads;
    unsigned int seed = time(NULL) * (childID + 1);

    for (int i = 0; i < points_per_thread; i++) {
        double x = (double)rand_r(&seed) / RAND_MAX;
        double y = (double)rand_r(&seed) / RAND_MAX;

        if (sqrt(x * x + y * y) <= 1) {
            points_inside_circle++;
        }
    }

    return NULL;
```

```
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        printf("Uso: %s <total_points> <total_threads>\n", argv[0]);
        return 1;
    }

    total_points = atoi(argv[1]);
    total_threads = atoi(argv[2]);

    if (total_threads > MAX_THREADS) {
        printf("El número de hilos no puede ser mayor a %d\n", MAX_THREADS);
        return 1;
    }

    if (total_points % total_threads != 0) {
        printf("El número de puntos debe ser divisible entre el número de hilos\n");
        return 1;
    }

    // Acá deberán introducir el paralelismo
    throw_darts((void*)1);

    double pi = 4.0 * points_inside_circle / total_points;
    printf("Valor de pi: %f\n", pi);

    return 0;
}
```

Modifiquen el código anterior para que el cálculo de $\pi$ se realice en paralelo utilizando pthreads. Recuerden que cada hilo debe generar una cantidad igual de puntos.

## Reporte de resultados

Deberán entregar un reporte con lo siguiente:

1. Implementación del programa en C, siguiendo las siguientes estrategias (un archivo por estrategia):
    1. El programa original, sin modificaciones.
    2. El programa modificado para que el cálculo de $\pi$ se realice en paralelo utilizando pthreads.
        1. No protejan la variable `points_inside_circle` con un mutex.
        2. Protejan la variable `points_inside_circle` con un mutex dentro del ciclo `for`.
        3. Protejan la variable `points_inside_circle` con un mutex fuera del ciclo `for`. En lugar de incrementar la variable global `points_inside_circle` dentro del ciclo, actualicen una variable local y al final del ciclo incrementen la variable global.
        4. Similar a 3, pero no actualicen `points_inside_circle` dentro de `throw_darts`. Retornen la cantidad de puntos dentro del círculo (usando un falso puntero) y hagan que el hilo principal incremente la variable global.
        5. Usen operaciones atómicas para incrementar la variable `points_inside_circle`.
2. Instrucciones detalladas de cómo compilar el programa. Si usaron un Makefile añádanlo.
3. Prueben su programa usando Helgrind o ThreadSanitizer y describan los problemas de concurrencia detectados, porqué sucedieron y cómo los corrigieron. Corríjanlos y vuelvan a probar.
    - Por supuesto, NO corrijan los errores en 1.2.1.
4. Realicen pruebas con diferentes cantidades de puntos y de hilos. Muestren los resultados obtenidos y el tiempo de ejecución en cada caso. Pueden usar el comando time para medir el tiempo de ejecución (por ejemplo, time ./pi 100000000 4).
5. ¿Cual implementación es más rápida? Consideren ejecutar su programa varias veces con diferentes cantidades

de puntos y de hilos.

1. Es recomendable que cierren todos los demás programas a la hora de realizar benchmarks, ya que esto puede afectar los resultados.
2. Los números exactos dependen de su máquina. Asegúrense de usar una cantidad de puntos alta para que todos los programas duren al menos unos segundos. En la cantidad de hilos, prueben con menos y más hilos que la cantidad de núcleos de su procesador. Por ejemplo:
    1. Puntos: 1000000000, Hilos: 1
    2. Puntos: 1000000000, Hilos: 4
    3. Puntos: 1000000000, Hilos: 8