1000 Tosses

Paralelo:

- 4 Threads:

CPU time: 0.000622 seconds

Wall time: 0 seconds

Error: 0.0215927

- 6 Threads:

CPU time: 0.001089 seconds

Wall time: 0 seconds

Error: 0.0215927

Sin threads:

Error: 0.0344073

CPU time: 0.000113 seconds

Wall time: 0 seconds

10000 Tosses

Paralelo:

- 4 Threads:

CPU time: 0.000476 seconds

Wall time: 0 seconds

Error: 0.0247927

- 6 Threads:

CPU time: 0.00096 seconds

Wall time: 0 seconds

Error: 0.0164073

Sin threads:

Error: 0.0196073

CPU time: 0.000405 seconds

Wall time: 0 seconds

100000 Tosses

Paralelo:

- 4 Threads:

CPU time: 0.001981 seconds

Wall time: 0 seconds

Error: 0.00239265

- 6 Threads:

CPU time: 0.001454 seconds

Wall time: 0 seconds

Error: 0.0111673

Sin threads:

Error: 0.00163265

CPU time: 0.00248 seconds

Wall time: 0 seconds

1000000 Tosses

Paralelo:

- 4 Threads:

CPU time: 0.024222 seconds

Wall time: 0 seconds

Error: 0.00326865

- 6 Threads:

CPU time: 0.023035 seconds

Wall time: 0 seconds

Error: 0.00179535

Sin threads:

Error: 0.00104865

CPU time: 0.033362 seconds

Wall time: 0 seconds

En general, el error variaba entre intentos, no era consistente dentro del mismo numero de tiros. Sin embargo, tanto para el programa que utilizaba hilos como el que no, el error disminuyó significativamente a medida que incrementaba el número de tiros.

En cuanto a el tiempo de ejecución, el código secuencial probó ser más eficiente que el paralelo en estimaciones con pocos tiros (1000 y 10000 tiros). Esto puede deberse a que la creación de hilos prueba ser más exigente que lo que se gana al paralelizar este algoritmo cuando la carga no es suficientemente alta. Por otro lado, para los intentos con más tiros (100000 y 1000000), el costo de la creación de hilos se vuelve negligible, entonces paralelizar los procesos sí prueba ser más eficiente.

En cuanto a la diferencia entre el uso de 4 hilos contra 6 hilos, vimos que a veces usar 6 era más rápido que 4, y otras veces usar 4 era mejor. La eficiencia de esto realmente depende de los recursos con los que se cuenta al momento de correr el programa. Es decir, si se tienen 4 cores disponibles, es más eficiente tener 4 hilos, ya que tener 6 llevaría a tener que cambiar procesos en ejecución, que puede ser más pesado. En general, el mejor rendimiento se tiene cuando el número de hilos es igual al número de cores disponibles.