**Alumno: Alonso Jesús Cerpa Salas.**

**Informe sobre el cálculo de Pi paralelo**

Datos de la computadora:

- 8 cores.

- Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8

Si se hace una suma de ‘n’ términos para hallar pi,

con n = 33,554,432 se obtienen los siguientes resultados:

|  |  |  |  |
| --- | --- | --- | --- |
| Threads\Tipo | Busy waiting NO OPTIMIZADO | Busy waiting OPTIMIZADO | Mutex |
| 1 | 339 ms | 109 ms | 109 ms |
| 2 | 2283 ms | 60 ms | 61 ms |
| 4 | 2529 ms | 45 ms | 46 ms |
| 8 | 2638 ms | 30 ms | 27 ms |
| 16 | - | 31 ms | 25 ms |
| 32 | - | 40 ms | 28 ms |
| 64 | - | 481 ms | 25 ms |
| 128 | - | 1920 ms | 25 ms |
| 256 | - | 6610 ms | 28 ms |

**Gráfico 1:**

Análisis del Gráfico 1: El método de busy waiting no optimizado es muy lento comparado con los otros, cuando tiene mas de un thread, y esto es porque en vez de paralelizar el cálculo de pi, el código es más que todo serial que se ejecuta repartido en varios threads, así, en vez de acelerar la ejecución la ralentiza. Esto se da cuando el código de la sección crítica es muy grande.

Por otro lado, ambos busy waiting optimizado y no optimizado sufren cuando el número de threads es mayor que el número de cores que posee la máquina, ya que hay momentos donde el contador del busy waiting no va a avanzar, no habrá ningún progreso, porque el thread que le toca ejecutar su sección crítica lo han mandado a dormir, se va a tener que esperar a que se despierte para poder avanzar, y además esto se puede dar varias veces.

Finalmente, el mutex evita el anterior problema porque ejecuta los threads solo uno a la vez en la sección crítica, pero sin importarle el orden, de esta manera no existe un thread específico que le toque su turno en la sección crítica, y por consiguiente los demás threads tengan que esperarlo, ese caso nunca se dará.

**Gráfico 2:**

Análisis del Gráfico 2: Es importante destacar la mejora que nos brinda el paralelismo como se puede ver en el Gráfico 2, sin embargo, como es intuido, esta mejora se limita al número de cores que poseamos en nuestra computadora, en este caso 8 cores, es por esto que el tiempo empieza a incrementarse (busy waiting) o mantenerse (mutex) cuando se ejecuta el código con más de 8 threads.