***Programación Concurrente***

**Informe del trabajo práctico, 2do cuatrimestre de 2019**

**Universidad Nacional de Quilmes**

**Integrantes del grupo:**

* Francisco Spinelli
* Nahuel Iglesias

**Descripción de las clases utilizadas:**

* **Main:** se encarga de inicializar todos los componentes necesarios para el programa. Recibe la lista de números de tipo BigInteger a analizar, y se encarga de iniciar el funcionamiento del resto de los componentes del programa. Al final, toma los números perfectos que fueron almacenados en el *Buffer* para imprimirlos junto con el tiempo de ejecución del programa.
* **Buffer:** se encarga de almacenar una parte de la lista de números de tipo BigInteger a analizar por los *PerfectWorkers*. Si se llena, y aún hay números para analizar, se espera a que se libere un espacio para cagar el próximo número. Una vez que ya no quedan números para analizar, es cargado con números negativos que indican a los *PerfectWorkers* que deben dejar de analizar. Luego, se encargan de guardar los números perfectos encontrados hasta que el *Main* los toma.
* **Barrier:** se encarga de impedir que el *Main* siga ejecutando funciones hasta que los *PerfectWorkers* hayan terminado de analizar todos los números existentes. Una vez que todos los *PerfectWorkers* finalizan sus funciones, libera al *Main* para que siga corriendo. Cuando el Buffer se empieza a llenar con los números perfectos encontrados, vuelve a detener al *Main* hasta que estén todos cargados.
* **PerfectWorker:** se encargan de analizar si los números existentes en el *Buffer* son números perfectos o no. Si intentan analizar un número negativo, detectan que deben dejar de analizar números y envían sus resultados al *Buffer*.
* **ThreadPool:** se encargan de crear e iniciar a los *PerfectWorkers*, y de cargar los números a analizar en el *Buffer*.

**Tiempos de ejecución con un Buffer de tamaño 10:**

* *Usando 5 PerfectWorkers:* tarda aproximadamente 130 ms en ejecutar.
* *Usando 10 PerfectWorkers:* tarda aproximadamente 120 ms en ejecutar.
* *Usando 15 PerfectWorkers:* tarda aproximadamente 103 ms en ejecutar.
* *Usando 20 PerfectWorkers:* tarda aproximadamente 105 ms en ejecutar.

**Tiempos de ejecución con un Buffer de tamaño 25:**

* *Usando 5 PerfectWorkers:* tarda aproximadamente 126 ms en ejecutar.
* *Usando 10 PerfectWorkers:* tarda aproximadamente 97 ms en ejecutar.
* *Usando 15 PerfectWorkers:* tarda aproximadamente 106 ms en ejecutar.
* *Usando 20 PerfectWorkers:* tarda aproximadamente 98 ms en ejecutar.

**Tiempos de ejecución con un Buffer de tamaño 50:**

* *Usando 5 PerfectWorkers:* tarda aproximadamente 108 ms en ejecutar.
* *Usando 10 PerfectWorkers*: tarda aproximadamente 98 ms en ejecutar.
* *Usando 15 PerfectWorkers:* tarda aproximadamente 103 ms en ejecutar.
* *Usando 20 PerfectWorkers:* tarda aproximadamente 101 ms en ejecutar.

**Conclusiones sobre los tiempos de ejecución en relación al tamaño del Buffer y la cantidad de Threads:**

Comparando los tiempos anteriores con una ejecución prácticamente secuencial de *200 ms* (es decir, usando un único PerfectWorker y un Buffer de tamaño 200), se puede ver una clara mejora utilizando threading, ya que cualquier ejecución con threading tiene tiempos menores que la ejecución secuencial.

Comparando las diversas ejecuciones utilizando threading con distintos recursos, se pueden ver dos cosas. Por un lado, cuanto mayor sea el tamaño del Buffer, menores van a ser los tiempos de ejecución generales. Por el otro, la cantidad más eficiente de PerfectWorkers depende del tamaño del Buffer (en el Buffer de tamaño 10 lo mejor es usar 15 PerfectWorkers, pero en el Buffer de 25 y el de 50 lo mejor es usar 10 PerfectWorkers).