



Programación Concurrente

1c 2022

Trabajo Práctico

Números Perfectos

Nombre	e-mail	Legajo
Malsam Leandro	leandro.malsam@gmail.com	43779
Soñez Mailin	mailinsonez@gmail.com	28708

Introducción

En el presente trabajo se pretende encontrar [números perfectos](#), es decir, números cuyos divisores propios sumados dan como resultado el mismo número. Y para ello el procesamiento se realizará de manera concurrente con varios hilos de ejecución.

Se diseñaron los objetos definidos en el [enunciado](#) provisto por la cátedra. Estos eran:

- **Main:** Se encarga de iniciar la ejecución y crear todos los demás objetos necesarios para el procesamiento. Además funciona como objeto “productor” generando los números secuenciales a ser evaluados. Estos números los va almacenando en un buffer, que funciona como un monitor.
- **PerfectWorker:** Se encarga de probar si un número (de los generados por el Main) es un número perfecto. Pueden existir 1 o más instancias de esta clase, cada una de las cuales se ejecuta como un thread independiente. Cada thread funciona como un “consumidor” leyendo números del buffer. Y a su vez, cuando encuentra un número perfecto, lo guarda en otro buffer de encontrados, que también es un monitor.
- **ThreadPool:** Este objeto tiene como tarea principal iniciar la ejecución de los threads *workers*, y también finalizarlos mediante el envío de una señal preestablecida (de acuerdo al enunciado, el número -1).
- **Buffer:** Implementación de un buffer de tamaño N como un monitor, que permite ser utilizado por varios threads de manera concurrente. Se utilizó un diseño simple como lo visto en la teoría, y se utilizaron 2 instancias, la primera para los números secuenciales que deben ser probados, y la segunda para los números perfectos que son encontrados.

No fue necesario diseñar objetos nuevos. Se agregaron tests unitarios que permitieron probar distintas partes individuales de la solución, y también plantear los escenarios finales con las combinaciones necesarias para analizar resultados.

La solución propuesta permite parametrizar el tamaño del buffer de números que se van generando, la cantidad de threads que procesarán dichos números, y la cantidad de números perfectos que se desean encontrar.

Evaluación

Inicialmente se realizó una prueba de concepto, implementando una solución rápida en Python, que encontró 5 números perfectos, ejecutando 1 solo thread, en unos 13847 segundos (aproximadamente 3 hs 50 min.). Esto nos permitió tener un valor de referencia sobre el cual comparar y determinar si nuestra solución en Java podría llegar a tardar demasiado.

Una vez terminada la solución, con una batería de tests unitarios, se plantearon los distintos escenarios como tests de la clase Main.

Los escenarios se ejecutaron en 4 equipos (3 físicos y 1 virtual) con las siguientes particularidades:

PC-1:

- Procesador: AMD Ryzen 5 3600 6-Core Processor 3.6GHz (6 cores - 12 threads)
- Memoria RAM: 16 GB
- SO: Windows 10 Pro 64 bits

Notebook-1:

- Procesador: Intel Core i5 4300M 2.6GHz (2 cores - 4 threads)
- Memoria RAM: 8 GB
- SO: Windows 10 Enterprise 64 bits

Notebook-2:

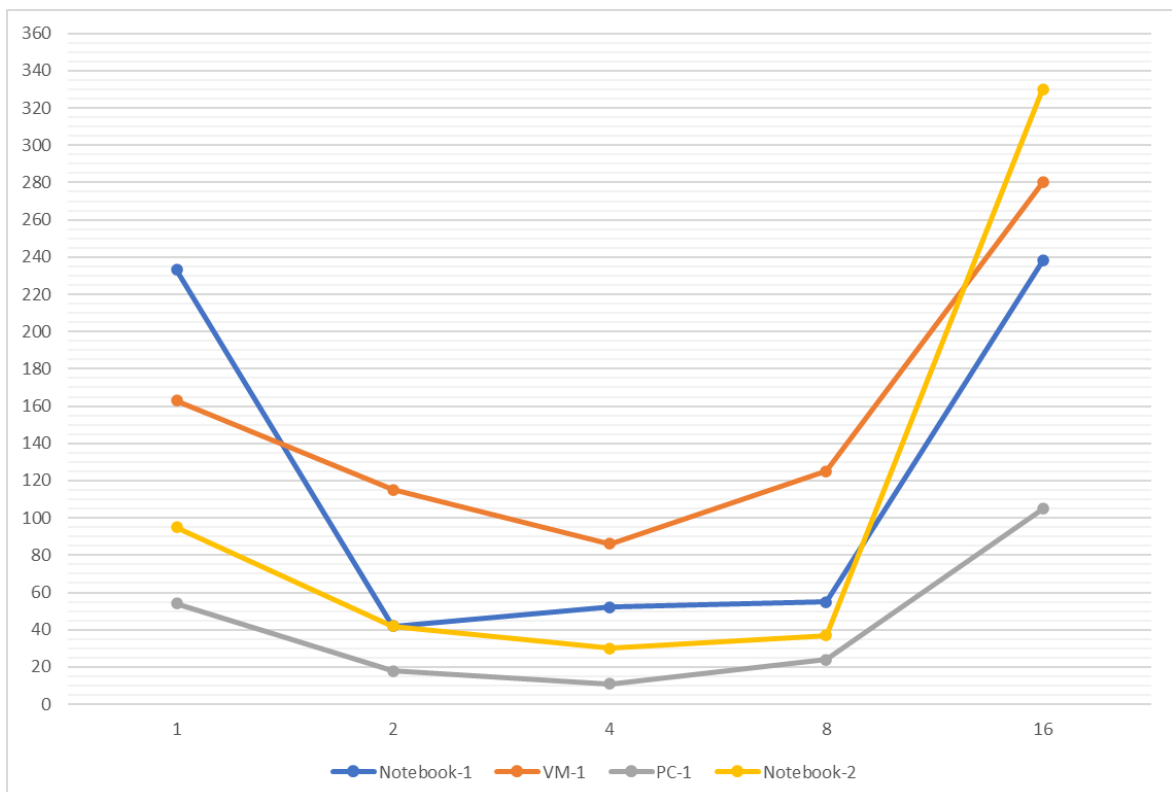
- Procesador: Intel Core i5 7200U 3.1GHz (2 cores - 4 threads)
- Memoria RAM: 8 GB
- SO: Ubuntu 20.04.4

VM-1:

- Procesador: Emulated Ryzen 5 2400G 3.6 GHz (4 cores - 8 threads)
- Memoria RAM: 8 GB
- SO: Linux Manjaro 5.4.195

Escenario: Buscando 4 números perfectos

Se utilizó únicamente un buffer de 16 posiciones, y se corrieron pruebas para 1, 2, 4, 8 y 16 threads.



Equipo	Threads	ms
Notebook-1	1	233
Notebook-1	2	42
Notebook-1	4	52
Notebook-1	8	55
Notebook-1	16	238

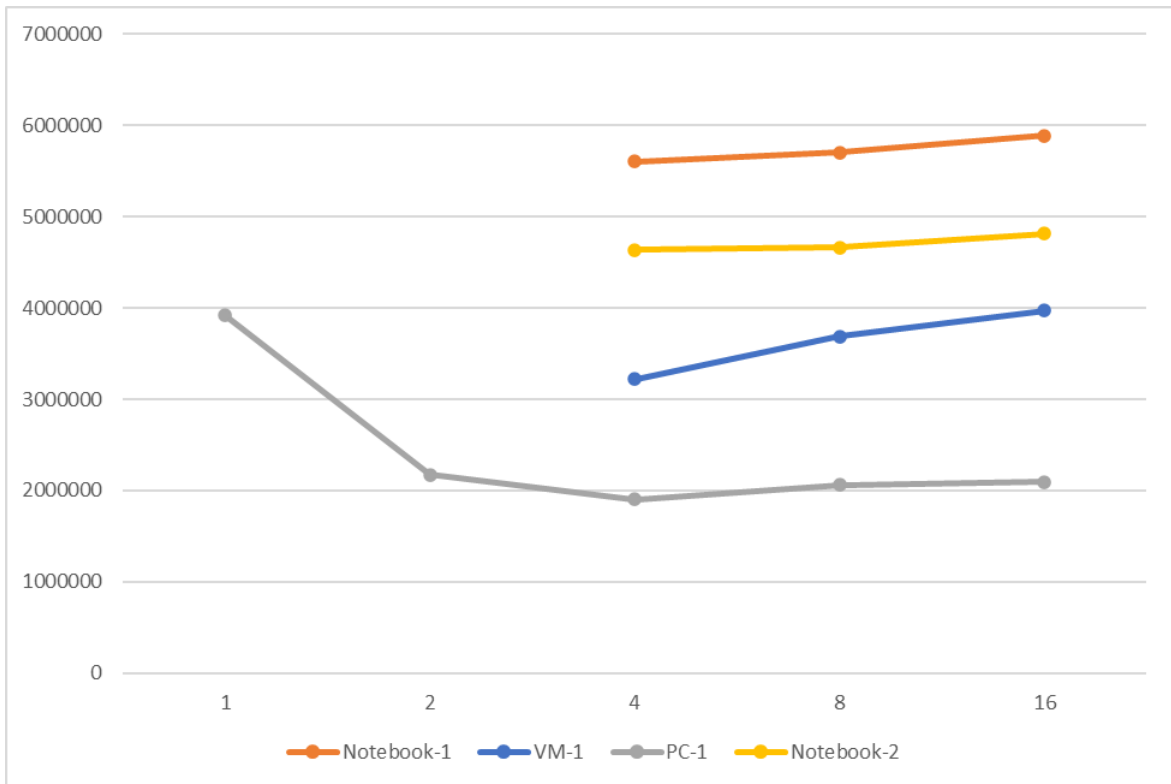
VM-1	1	163
VM-1	2	115
VM-1	4	86
VM-1	8	125
VM-1	16	280

PC-1	1	54
PC-1	2	18
PC-1	4	11
PC-1	8	24
PC-1	16	105

Notebook-2	1	95
Notebook-2	2	42
Notebook-2	4	30
Notebook-2	8	37
Notebook-2	16	330

Escenario: Buscando 5 números perfectos

Se utilizó únicamente un buffer de 16 posiciones, y se corrieron pruebas para 4, 8 y 16 threads. En el equipo más rápido (PC-1) también se probó con 1 y 2 threads.



Equipo	Threads	ms
Notebook-1	4	5603274
Notebook-1	8	5704086
Notebook-1	16	5892030

VM-1	4	3220111
VM-1	8	3686000
VM-1	16	3968736

PC-1	1	3918426
PC-1	2	2171107
PC-1	4	1904264
PC-1	8	2058891
PC-1	16	2094367

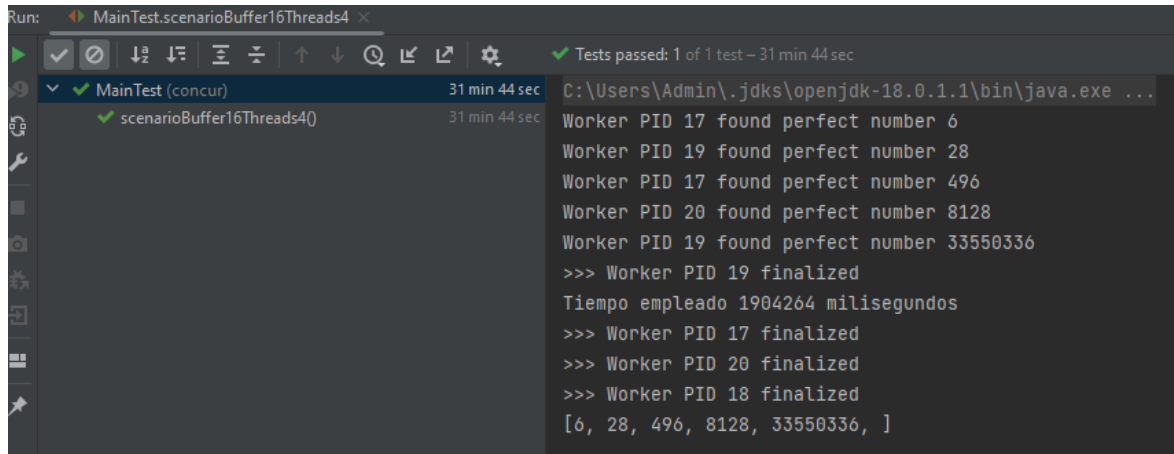
Notebook-2	4	4635144
Notebook-2	8	4661139
Notebook-2	16	4815638

Análisis

Una vez realizadas todas las evaluaciones en ambos escenarios, se llegó a la conclusión de que en todos los casos, el uso de 4 threads combinado con buffer de 16 posiciones fue el más eficiente.

Para la búsqueda de 5 números perfectos, obtuvimos como **mejor resultado** un tiempo de 31min 44seg (1904264ms) en la PC-1.

Mejor resultado: PC-1 - 4 threads - buffer de 16 posiciones



```
Run: MainTest.scenarioBuffer16Threads4
Tests passed: 1 of 1 test - 31 min 44 sec

MainTest (concur) 31 min 44 sec
  scenarioBuffer16Threads4() 31 min 44 sec

C:\Users\Admin\.jdk\openjdk-18.0.1.1\bin\java.exe ...
Worker PID 17 found perfect number 6
Worker PID 19 found perfect number 28
Worker PID 17 found perfect number 496
Worker PID 20 found perfect number 8128
Worker PID 19 found perfect number 33550336
>>> Worker PID 19 finalized
Tiempo empleado 1904264 milisegundos
>>> Worker PID 17 finalized
>>> Worker PID 20 finalized
>>> Worker PID 18 finalized
[6, 28, 496, 8128, 33550336, ]
```

El **peor resultado**, para la búsqueda de 5 números, se obtuvo con la Notebook-1 ejecutando 16 threads. Empleó 1h 38min 12seg (5892030 ms).

Para la búsqueda de 4 números, todas las pruebas dieron valores aproximados, con diferencias imperceptibles para el humano, pero fácilmente distinguibles para una computadora. Las curvas en forma de parábola son similares para todos los equipos.

Para la búsqueda de 5 números, no se obtuvieron resultados con 1 o 2 threads, porque el procesamiento tardó demasiado y se optó por cortar la ejecución. Para el resto de las pruebas, se ve que los equipos arrojan curvas similares en cuanto a su forma, pero con diferencias numéricas bien marcadas. En estas pruebas llama la atención que el equipo virtualizado (VM-1) arrojó mejores resultados que las notebooks. Esto, intuimos, se debe a que dicho equipo virtual tiene un mejor procesador.

En todos los casos, se observa que utilizar un solo thread no es lo ideal. En estos casos el procesamiento se hace siempre en forma secuencial y se desperdicia poder de procesamiento.

Y en el otro extremo, las pruebas con 16 threads también muestran que los tiempos de ejecución aumentan. Esto se debe a que hay demasiados threads, que superan a la capacidad del procesador, por lo que se produce una mayor cantidad de *context switch*. La gestión que tiene que hacer el SO es mayor por lo que se dedica tiempo de procesamiento a estas tareas “administrativas”.

Para finalizar, concluimos que los mejores resultados se obtienen cuando la cantidad de threads se acerca a la capacidad del procesador. Pocos threads desperdician poder de cómputo, mientras que muchos threads producen un overhead contraproducente.