

**Escuela Colombiana De Ingeniería
Julio Garavito**

Javier Ivan Toquica Barrera

Juan Daniel Murcia Sanchez

Juan David Parroquiano Roldan

Andres Felipe Montes Ortiz

Arquitecturas de software

Laboratorio No.1

2024-2

Introducción

En la programación moderna, la capacidad de realizar múltiples tareas de manera simultánea es esencial para aprovechar al máximo el poder de procesamiento de los sistemas multiprocesador. El paralelismo, que permite la ejecución de múltiples hilos de manera concurrente, es una técnica fundamental para mejorar el rendimiento de las aplicaciones, especialmente en entornos donde se manejan grandes volúmenes de datos o tareas complejas.

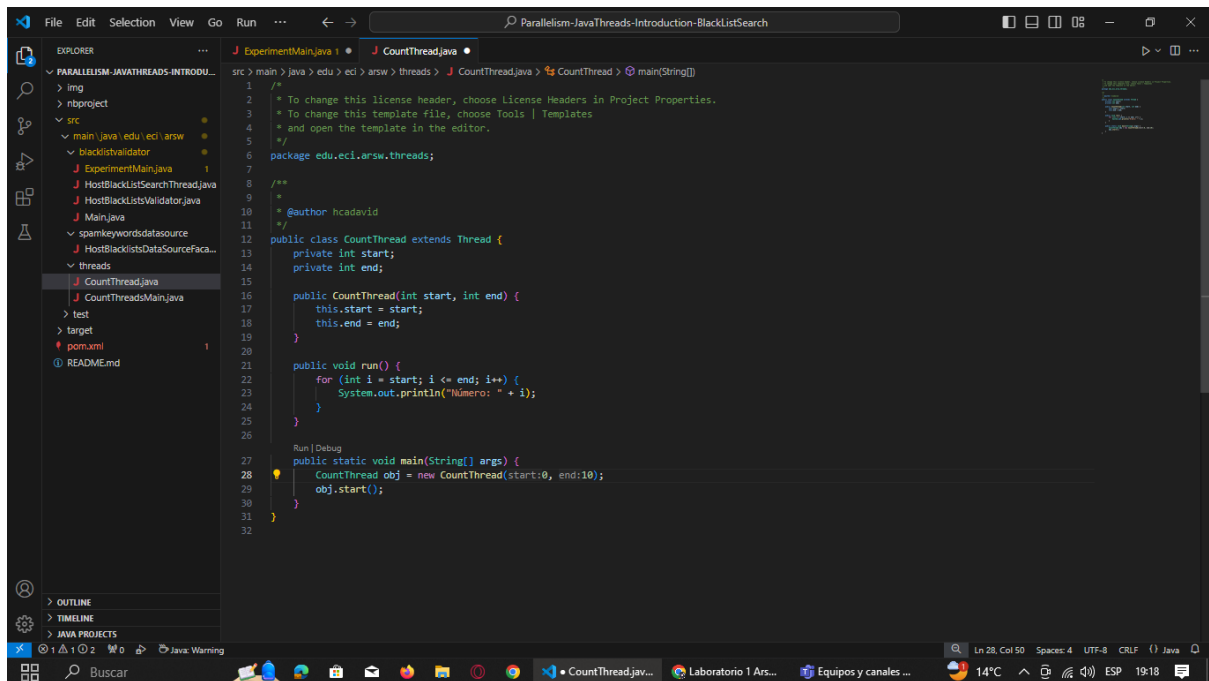
Este laboratorio se enfoca en la implementación y comprensión de hilos en Java, un lenguaje ampliamente utilizado que ofrece una robusta API para la gestión de concurrencia. A lo largo del ejercicio, exploraremos la creación y el manejo de hilos, así como la optimización de tareas a través del paralelismo. El objetivo es comprender cómo dividir un problema en subproblemas independientes que puedan ser resueltos en paralelo, y cómo sincronizar los resultados para obtener una solución eficiente.

En la primera parte, se abordarán los conceptos básicos de hilos en Java, implementando clases que permitan la ejecución concurrente de diferentes tareas. La segunda parte se centra en un caso de estudio real, donde se desarrolla un componente para la validación de direcciones IP en listas negras. Este ejercicio no solo permitirá aplicar los conocimientos adquiridos, sino que también ofrecerá una visión práctica de cómo el paralelismo puede acelerar procesos complejos en aplicaciones de seguridad informática.

Finalmente, se evaluará el rendimiento de la solución mediante una serie de experimentos que medirán el tiempo de ejecución y el uso de recursos en diferentes escenarios de concurrencia. Este análisis permitirá reflexionar sobre las limitaciones y ventajas del paralelismo en la programación, proporcionando una base sólida para el desarrollo de aplicaciones más eficientes y escalables en el futuro.

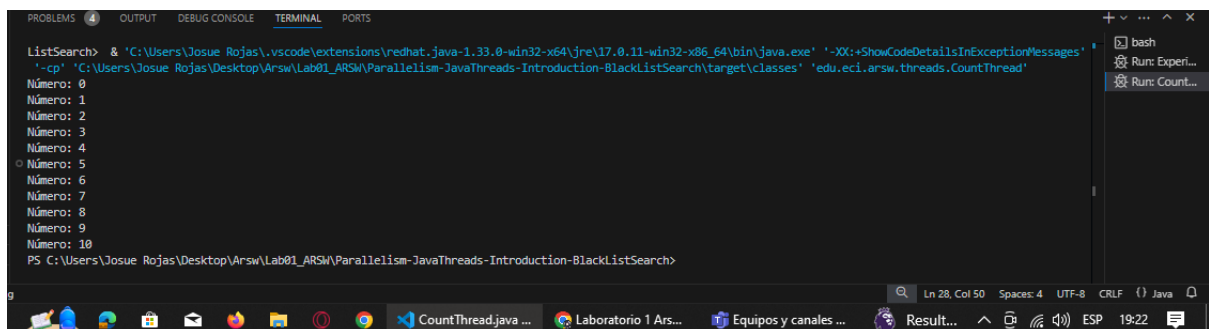
Parte I Hilos Java

De acuerdo con lo revisado en las lecturas, complete las clases `CountThread`, para que las mismas definan el ciclo de vida de un hilo que imprima por pantalla los números entre A y B.



```
src > main > java > edu > ed > arsw > threads > J CountThread.java > CountThread > main(String[])
1  /**
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package edu.eci.arsw.threads;
7
8  /**
9  *
10  * @author hcadavid
11  */
12  public class CountThread extends Thread {
13      private int start;
14      private int end;
15
16      public CountThread(int start, int end) {
17          this.start = start;
18          this.end = end;
19      }
20
21      public void run() {
22          for (int i = start; i <= end; i++) {
23              System.out.println("Número: " + i);
24          }
25      }
26
27      Run | Debug
28      public static void main(String[] args) {
29          CountThread obj = new CountThread(start:0, end:10);
30          obj.start();
31      }
32  }
```

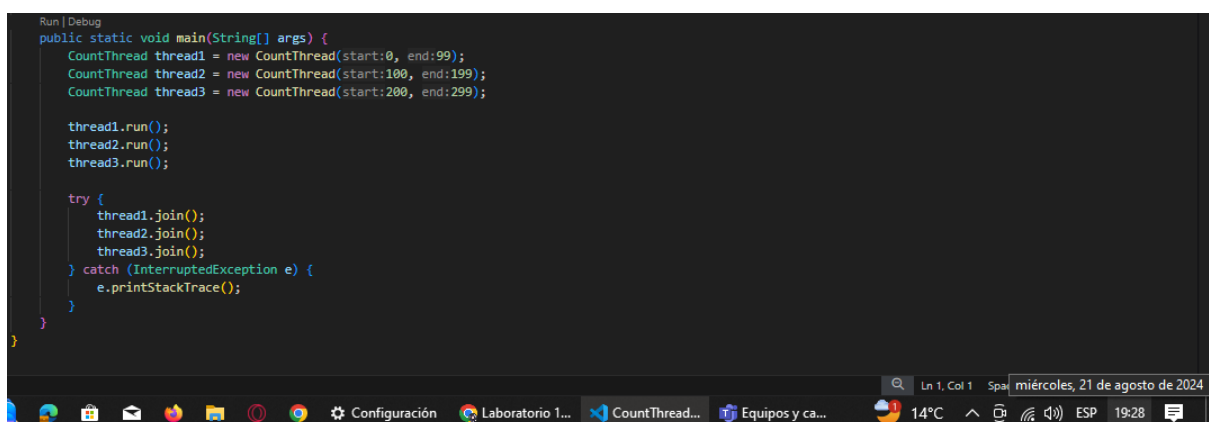
Para terminar este punto recurrimos a extender de la clase Thread y sobrescribir el método run() para que imprima por pantalla los valores de A hasta B



```
PS C:\Users\Josue_Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlackListSearch>
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Número: 10
PS C:\Users\Josue_Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlackListSearch>
```

Complete el método main de la clase CountMainThreads para que:

- Cree 3 hilos de tipo CountThread, asignándole al primero el intervalo [0..99], al segundo [99..199], y al tercero [200..299].



```
Run | Debug
public static void main(String[] args) {
    CountThread thread1 = new CountThread(start:0, end:99);
    CountThread thread2 = new CountThread(start:100, end:199);
    CountThread thread3 = new CountThread(start:200, end:299);

    thread1.run();
    thread2.run();
    thread3.run();

    try {
        thread1.join();
        thread2.join();
        thread3.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

- Inicie los tres hilos con 'start()'.

```
21 public static void main(String[] args) {
22     CountThread thread1 = new CountThread(start:0, end:99);
23     CountThread thread2 = new CountThread(start:100, end:199);
24     CountThread thread3 = new CountThread(start:200, end:299);
25
26     thread1.start();
27     thread2.start();
28     thread3.start();
29 }
```

659-54d6-417d-ac42-8695168e0619Número: 100
Número: 101
Número: 0
Número: 200
Número: 1
Número: 102
Número: 103
Número: 2
Número: 201
Número: 202
Número: 3
Número: 4
Número: 104
Número: 5
Número: 203
Número: 204
Número: 105
Número: 205
Número: 6
Número: 7
Número: 106
Número: 8
Número: 206
Número: 9

Ln 28, Col 22 Spa miércoles, 21 de agosto de 2024

- Ejecute y revise la salida por pantalla.

Número: 194
Número: 84
Número: 85
Número: 195
Número: 86
Número: 196
Número: 197
Número: 198
Número: 199
Número: 87
Número: 88
Número: 89
Número: 90
Número: 91
Número: 92
Número: 93
Número: 94
Número: 95
Número: 96
Número: 97
Número: 98
Número: 99

PS C:\Users\Josue_Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlacklistSearch>

Ln 28, Col 22 Spa miércoles, 21 de agosto de 2024

- Cambie el incio con 'start()' por 'run()'. Cómo cambia la salida?, por qué?.

```
21 public static void main(String[] args) {
22     CountThread thread1 = new CountThread(start:0, end:99);
23     CountThread thread2 = new CountThread(start:100, end:199);
24     CountThread thread3 = new CountThread(start:200, end:299);
25
26     thread1.run();
27     thread2.run();
28     thread3.run();
29 }
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

rsw.threads.CountThreadsMain'

659-54d6-417d-ac42-8695168e0619Número: 0

Número: 1

Número: 2

Número: 3

Número: 4

Número: 5

Número: 6

Número: 7

Número: 8

Número: 9

Número: 10

Número: 11

Número: 12

Número: 13

Número: 14

Número: 15

Número: 16

Número: 17

Número: 18

Número: 19

Número: 20

Número: 21

Número: 22

Ln 26, Col 20 Spa miércoles, 21 de agosto de 2024

Cuando usamos el método `run()` se ejecuta en el mismo hilo por lo tanto hasta que no termina de ejecutar su ciclo de vida no pasa a la siguiente línea (es secuencial), mientras que al usar el método `start()` se crea un nuevo hilo y ahí se ejecuta el `run()` por lo tanto ya no es secuencial si no paralelo

Parte II Hilos Java

Cree una clase de tipo `Thread` que represente el ciclo de vida de un hilo que haga la búsqueda de un segmento del conjunto de servidores disponibles. Agregue a dicha clase un método que permita 'preguntarle' a las instancias del mismo (los hilos) cuantas ocurrencias de servidores maliciosos ha encontrado o encontró.

```
File Edit Selection View Go Run ... Parallelism-JavaThreads-Introduction-BlackListSearch
```

EXPLORER

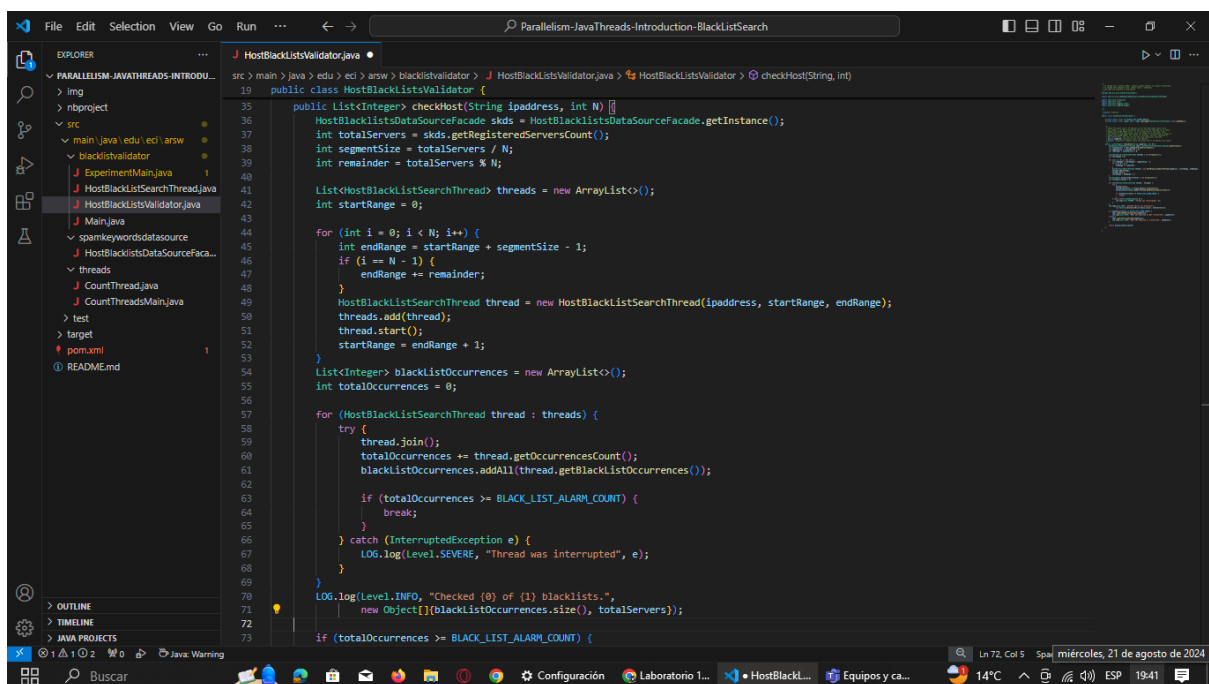
- PARALLELISM-JAVATHREADS-INTRODU...
- img
- nbproject
- src
 - main (java \ edu \ eci \ arsw)
 - blacklistvalidator
 - ExperimentMain.java
 - HostBlackListSearchThread.java
 - HostBlackListValidator.java
 - Main.java
 - spamkeywordsdatasource
 - HostBlacklistsDataSourceFaca...
 - threads
 - test
 - target
 - pom.xml
 - README.md

```
src > main > java > edu > eci > arsw > blacklistvalidator > HostBlackListSearchThread.java > HostBlackListSearchThread > getOccurrencesCount()
```

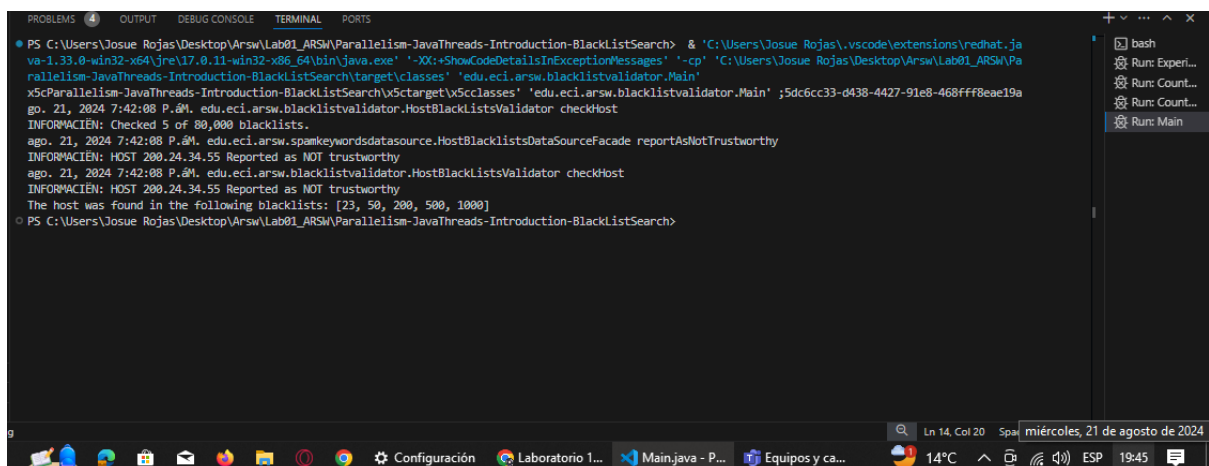
```
1 package edu.eci.arsw.blacklistvalidator;
2
3 import edu.eci.arsw.spamkeywordsdatasource.HostBlacklistsDataSourceFacade;
4 import java.util.LinkedList;
5 import java.util.List;
6
7 public class HostBlackListSearchThread extends Thread {
8
9     private final HostBlacklistsDataSourceFacade skds;
10    private final String ipaddress;
11    private final int startRange;
12    private final int endRange;
13    private int occurrencesCount;
14    private final List<Integer> blacklistOccurrences;
15
16    public HostBlackListSearchThread(String ipaddress, int startRange, int endRange) {
17        this.skds = HostBlacklistsDataSourceFacade.getInstance();
18        this.ipaddress = ipaddress;
19        this.startRange = startRange;
20        this.endRange = endRange;
21        this.occurrencesCount = 0;
22        this.blacklistOccurrences = new LinkedList<>();
23    }
24    @Override
25    public void run() {
26        for (int i = startRange; i <= endRange; i++) {
27            if (skds.isInBlackListServer(i, ipaddress)) {
28                blacklistOccurrences.add(i);
29                occurrencesCount++;
30            }
31        }
32    }
33    public int getOccurrencesCount() {
34        return occurrencesCount;
35    }
36    public List<Integer> getBlacklistOccurrences() {
37        return blacklistOccurrences;
38    }
39
40 }
```

Ln 35, Col 8 Spa miércoles, 21 de agosto de 2024

Agregue al método 'checkHost' un parámetro entero N, correspondiente al número de hilos entre los que se va a realizar la búsqueda (recuerde tener en cuenta si N es par o impar!). Modifique el código de este método para que divida el espacio de búsqueda entre las N partes indicadas, y paralelice la búsqueda a través de N hilos. Haga que dicha función espere hasta que los N hilos terminen de resolver su respectivo sub-problema, agregue las ocurrencias encontradas por cada hilo a la lista que retorna el método, y entonces calcule (sumando el total de ocurrencias encontradas por cada hilo) si el número de ocurrencias es mayor o igual a BLACK_LIST_ALARM_COUNT. Si se da este caso, al final se DEBE reportar el host como confiable o no confiable, y mostrar el listado con los números de las listas negras respectivas. Para lograr este comportamiento de 'espera' revise el método join del API de concurrencia de Java. Tenga también en cuenta:



```
src > main > java > edu > arsw > blacklistvalidator > HostBlackListsValidator.java > HostBlackListsValidator > checkHost(String, int)
19 public class HostBlackListsValidator {
35
36 public List<Integer> checkHost(String ipAddress, int N) {
37     HostBlackListsDataSourceFacade sdfs = HostBlackListsDataSourceFacade.getInstance();
38     int totalServers = sdfs.getRegisteredServersCount();
39     int segmentSize = totalServers / N;
40     int remainder = totalServers % N;
41
42     List<HostBlackListSearchThread> threads = new ArrayList<>();
43     int startRange = 0;
44     for (int i = 0; i < N; i++) {
45         int endRange = startRange + segmentSize - 1;
46         if (i == N - 1) {
47             endRange += remainder;
48         }
49         HostBlackListSearchThread thread = new HostBlackListSearchThread(ipAddress, startRange, endRange);
50         threads.add(thread);
51         thread.start();
52         startRange = endRange + 1;
53     }
54     List<Integer> blacklistOccurrences = new ArrayList<>();
55     int totalOccurrences = 0;
56     for (HostBlackListSearchThread thread : threads) {
57         try {
58             thread.join();
59             totalOccurrences += thread.getOccurrencesCount();
60             blacklistOccurrences.add(thread.getBlackListOccurrences());
61             if (totalOccurrences >= BLACK_LIST_ALARM_COUNT) {
62                 break;
63             }
64         } catch (InterruptedException e) {
65             LOG.log(Level.SEVERE, "Thread was interrupted", e);
66         }
67     }
68     LOG.log(Level.INFO, "Checked {0} of {1} blacklists.",
69         new Object[] {blacklistOccurrences.size(), totalServers});
70     if (totalOccurrences >= BLACK_LIST_ALARM_COUNT) {
```



```
PS C:\Users\Josue Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlacklistSearch> & 'C:\Users\Josue Rojas\.vscode\extensions\redhat.java-1.33.0-win32-x64\jre\17.0.11-win32-x86_64\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Josue Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlacklistSearch\target\classes' 'edu.eci.arsw.blacklistvalidator.Main'
go. 21, 2024 7:42:08 P.M. edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFORMACIÖN: Checked 5 of 80,000 blacklists.
INFORMACIÖN: HOST 200.24.34.55 Reported as NOT trustworthy
go. 21, 2024 7:42:08 P.M. edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFORMACIÖN: HOST 200.24.34.55 Reported as NOT trustworthy
The host was found in the following blacklists: [23, 50, 200, 500, 1000]
PS C:\Users\Josue Rojas\Desktop\Arsw\Lab01_ARSW\Parallelism-JavaThreads-Introduction-BlacklistSearch>
```

Parte III Evaluación de Desempeño

A partir de lo anterior, implemente la siguiente secuencia de experimentos para realizar la validación de direcciones IP dispersas (por ejemplo 202.24.34.55), tomando los tiempos de ejecución de los mismos (asegúrese de hacerlos en la misma máquina):

- Un solo hilo.

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.6\lib\idea_rt.jar=57416:C:\Program Files\
Aug 22, 2024 2:13:49 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
Aug 22, 2024 2:13:49 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 2:13:49 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 1 threads:
Time taken: 118469 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Aug 22, 2024 2:14:04 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
Aug 22, 2024 2:14:04 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
```

- Tantos hilos como núcleos de procesamiento (haga que el programa determine esto haciendo uso del API Runtime).

```
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Aug 22, 2024 2:14:04 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
Aug 22, 2024 2:14:04 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 2:14:04 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 8 threads:
Time taken: 14567 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Aug 22, 2024 2:14:11 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
```

- Tantos hilos como el doble de núcleos de procesamiento.

```
-----
Aug 22, 2024 2:14:11 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
Aug 22, 2024 2:14:11 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 2:14:11 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 16 threads:
Time taken: 7382 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Aug 22, 2024 2:14:14 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blackLists.
Aug 22, 2024 2:14:14 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
```

- 50 hilos.

```
-----
Aug 22, 2024 2:14:14 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blacklists.
Aug 22, 2024 2:14:14 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 2:14:14 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 50 threads:
Time taken: 2319 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Aug 22, 2024 2:14:15 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
-----
```

- 100 hilos.

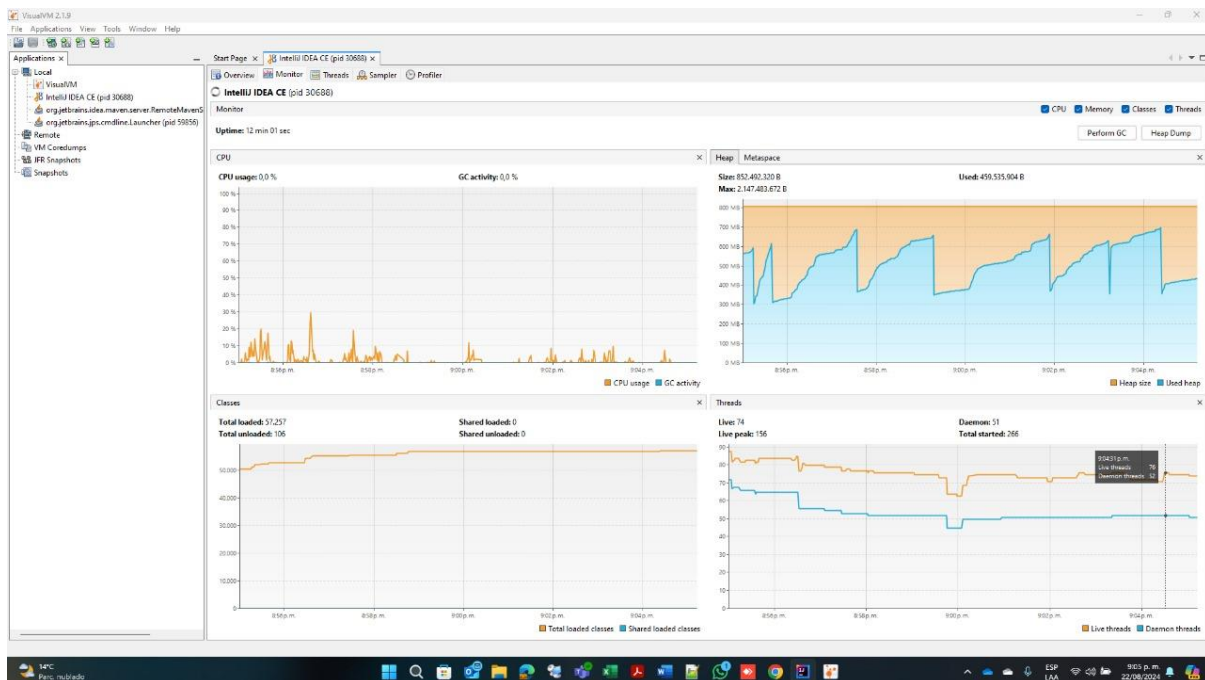
```
-----
Aug 22, 2024 2:14:15 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blacklists.
Aug 22, 2024 2:14:15 PM edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 2:14:15 PM edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 100 threads:
Time taken: 1152 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Process finished with exit code 0
-----
```

Al iniciar el programa ejecute el monitor jVisualVM, y a medida que corran las pruebas, revise y anote el consumo de CPU y de memoria en cada caso.

Para el caso con 20 hilos estos fueron los datos obtenidos para el uso de la CPU, memoria y el aumento en el número de hilos.

```
Project: Parallelism-JavaThreads-Introduction-BlackListSearch
src > main > java > edu > eci > arsw > blacklistvalidator > ExperimentMain
Run: ExperimentMain
C:\Users\JuanDavidParroquiano\jdk\openjdk-22.0.2\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.3\lib\idea_rt.jar=51618:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Aug 22, 2024 9:02:44 P. M. edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: Checked 5 of 80,000 blacklists.
Aug 22, 2024 9:02:44 P. M. edu.eci.arsw.spamkeywordsdatasource.HostBlackListsDataSourceFacade reportAsNotTrustworthy
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Aug 22, 2024 9:02:44 P. M. edu.eci.arsw.blacklistvalidator.HostBlackListsValidator checkHost
INFO: HOST 202.24.34.55 Reported as NOT trustworthy
Experiment with 20 threads:
Time taken: 6227 ms
Blacklists found: 5
Blacklist servers: [29, 10034, 20200, 31000, 70500]
-----
Process finished with exit code 0
```


Se puede evidenciar como al ejecutar el programa el uso de la CPU pasa de prácticamente 0 a tener más relevancia en el proceso, a pesar de todo esto y ya que se tomó una prueba pequeña (20 hilos) el uso del procesador y la memoria disponible para el programa fue mínimo.

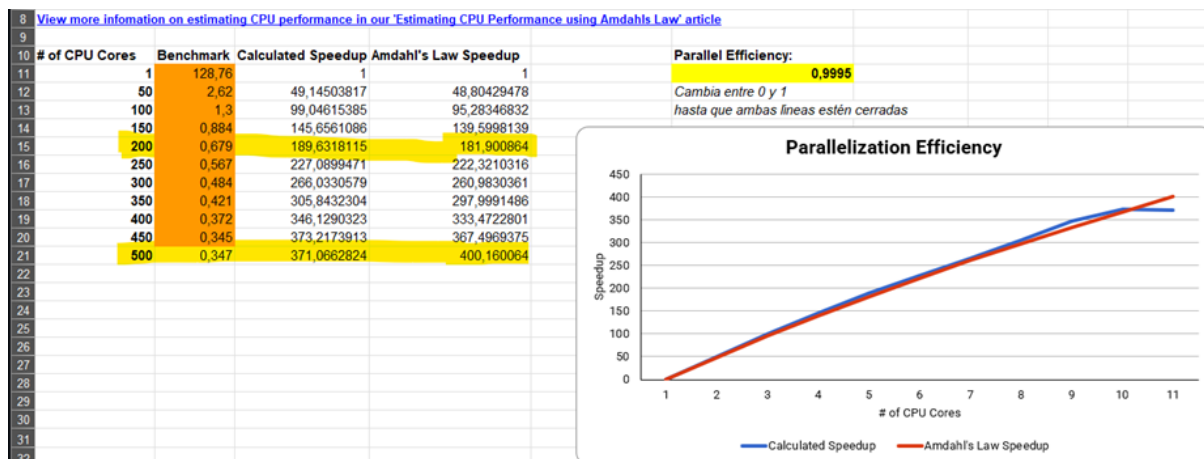


1. Según la [ley de Amdahls](#):

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

donde $S(n)$ es el mejoramiento teórico del desempeño, P la fracción paralelizable del algoritmo, y n el número de hilos, a mayor n , mayor debería ser dicha mejora. Por qué el mejor desempeño no se logra con los 500 hilos?, cómo se compara este desempeño cuando se usan 200?.

Los siguientes datos muestra los tiempos en segundos que tardó la ejecución del programa según el número de hilos (columna naranja). Podemos ver que el mejor desempeño se logra con 500 hilos.



La ley de Amdahl se expresa como:

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

Donde:

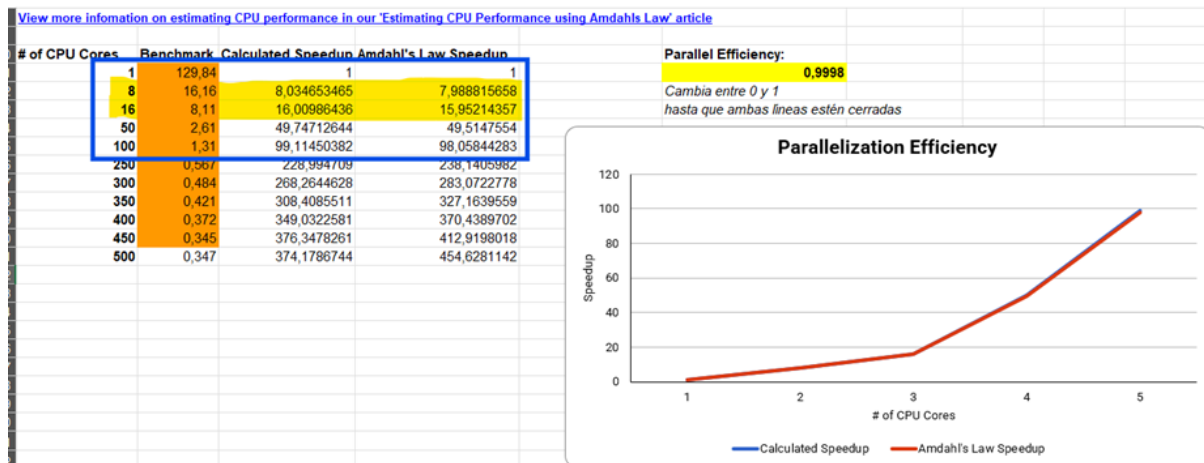
- $S(n)$: Es el mejoramiento teórico del desempeño al utilizar 'n' unidades de procesamiento (en este caso, hilos).
- P : Es la fracción del trabajo que puede ser paralelizada.
- n : Es el número de hilos utilizados.

A medida que aumentamos el número de hilos, se espera una mejora en el rendimiento. Sin embargo, la Ley de Amdahl muestra que esta mejora no es lineal y tiene un límite, debido a la parte del código que no se puede paralelizar.

El rendimiento mejora a medida que agregas más hilos, pero después de un cierto punto, agregar más hilos solo resulta en una pequeña mejora adicional. Esto se debe a la fracción $(1 - P)$ que no se puede paralelizar. Incluso si usas 500 hilos, esa parte del proceso siempre se ejecutará de manera secuencial, limitando la aceleración total.

2. Cómo se comporta la solución usando tantos hilos de procesamiento como núcleos comparado con el resultado de usar el doble de éste?.

De acuerdo con nuestra gráfica, observamos que al duplicar el número de núcleos a 16, el desempeño mejoró proporcionalmente, logrando un rendimiento dos veces superior al inicial con 8 núcleos.



3. De acuerdo con lo anterior, si para este problema en lugar de 100 hilos en una sola CPU se pudiera usar 1 hilo en cada una de 100 máquinas hipotéticas, la ley de Amdahl se aplicaría mejor?. Si en lugar de esto se usaran c hilos en 100/c máquinas distribuidas (siendo c es el número de núcleos de dichas máquinas), se mejoraría?. Explique su respuesta.

Usar 1 hilo en cada una de las 100 máquinas hipotéticas:

Teóricamente, la Ley de Amdahl se aplica mejor aquí porque cada máquina ejecuta su hilo de forma independiente, reduciendo la contención de recursos. No obstante, el rendimiento real también dependerá de la eficiencia de la comunicación y sincronización entre las máquinas.

Usar c hilos en 100/c máquinas distribuidas:

Si se optimiza bien la ejecución paralela dentro de cada máquina y se minimizan las sobrecargas de sincronización, es probable que se obtenga una mejora en el rendimiento. Sin embargo, como en el primer escenario, la eficiencia de la comunicación entre máquinas sigue siendo un factor crucial.