Universidad de Buenos Aires
Facultad de Ingeniería

Técnicas de Programación Concurrente I
Catedra Deymonnaz

Trabajo Práctico N.º 1: “Fork Join”

2do Cuatrimestre 2025

Alumno:

- Castro, Martín (109807)

Castro, Martin - 109807

Introducción

El objetivo de este trabajo práctico fue aplicar el modelo de **Fork-Join** utilizando la librería **Rayon** en Rust para procesar un archivo de gran tamaño y evaluar el impacto de la paralelización en la performance. Se buscó un dataset con un volumen considerable que permitiera observar de manera clara las diferencias de tiempo y uso de recursos al ejecutar el programa con distinta cantidad de hilos.

Dataset (subtitulo)

Para el desarrollo del tp se utilizó el siguiente dataset de Kaggle, se eligió usar el csv *male_players_23.csv* (5.64 GB), el cual contiene las distintas cartas de los jugadores masculinos del videojuego FIFA 23.

La interpretación que se hizo sobre las columnas de csv fueron (solo vamos a enumerar aquellas que fueron relevantes para la consulta):

- *id_player*: Numero de identificación unica del jugador
- *name*: Nombre corto del jugador o apodo (Ejemplo: L. Messi)
- *team*: Equipo en el que juega actualmente
- *position*: Posiciones que puede ocupar en el videojuego (Ejemplo: “CM”, “ST”, etc)
- *country_name*: Nombre del país al que representa
- *overall*: Valoracion del jugador
- *height_cm*: Altura del jugador en centímetros
- *weight_cm*: Peso del jugador en kilos

Transformaciones aplicadas (titulo)

Como parte de un pre-procesamiento se dividió el dataset en 4 partes para poder procesarlo de manera eficiente.

Al explorar el csv se observó que algunas columnas eran incómodas de parsear ya que tenían el formato *“atributo_1, …. , atributo_n”* por lo que se decidió utilizar el crate *csv*. Además, de *csv* los otros crates que se usaron fueron *json* y *serde_json* para facilitarles la salida (se imitó la salida pedida para tps de cuatrimestres anteriores), *sysinfo* para medir el uso de memoria y por último *Rayon* para poder trabajar con el modelo de fork join requerido.

Las consultas realizadas sobre el csv fueron:

- Top 10 nacionalidades con más jugadores. Para cada nacionalidad (en caso de empate resolver alfabéticamente):
    - Porcentaje del total de jugadores (2 decimales)
    - promedio del overall (redondeado a 2 decimales).

- Top 10 de mejores jugadores ordenados por la suma de todos sus atributos (en caso de empate resolver alfabéticamente)
    - De cada jugador el promedio de los atributos, el overall, el equipo en el que juegan, la posición y el país que representan
    - El IMC de ese jugador

Instrucciones para ejecutar el programa (subtítulo)

1) Se debe descargar el csv mencionado
2) Se debe correr el programa dentro de la carpeta *split* para particionar el csv

Una vez hecho esto para correr el programa (en la carpeta tp) se hace de la siguiente manera:

cargo run –release <cant_threads> <cant_de_ejecuciones>

Análisis de Performance (titulo)

Se realizaron 100 corridas del programa y estos fueron los resultados promedio:

Threads	Tiempo (s)	Memoria (GB)
4	7.04	9.92
2	14.74	10.14
1	22.75	10.88

Se observa que al aumentar la cantidad de threads, el tiempo de ejecución disminuye significativamente, esto muestra que el programa se beneficia de la paralelización, ya que al tener 4 threads se están leyendo los 4 archivos a la vez, mientras que con 2 y con 1 este proceso es más lento (especialmente con 1).

Por otra parte, podemos observar que el comportamiento de la memoria no es lineal:

- Con un thread, la memoria fue mayor debido a que un solo proceso concentró la totalidad de las estructuras en RAM.

- Con 2 threads, el consumo disminuyó levemente, al distribuirse las cargas.

- Con 4 threads, la memoria volvió a crecer, ya que cada hilo mantenía estructuras propias en paralelo, elevando el uso total.

Este alto consumo de memoria puede deberse al tipo de dato que estamos usando (HashMap) estos son más costosos que otros tipos como los *vec*.

Se hicieron pruebas de rendimiento reemplazando la estructura HashMap por vec pero había una pérdida de performance en el tiempo de ejecución ya que para las consultas planteadas no podemos quedarnos con elementos repetidos (cada jugador suele tener 2 o 3 cartas en un mismo videojuego de esta franquicia) por lo que el costo a la larga era mayor y se prefirió tener una mayor velocidad de ejecución a cambio de perder performance en memoria.

Conclusiones y posibles mejoras

Como se pudo ver en el desarrollo la paralelización mediante el método de Fork Join con Rayon mejora notablemente el tiempo de ejecución. Aunque también vimos que hay un **trade-off** entre velocidad y consumo de memoria.

Algunas posibles mejoras pueden ser explorar otro tipo de dato que permita tener un mayor rendimiento en la etapa de *reduce* y procesamiento, fijar una cantidad de chunks a leer en vez de que cada thread lea un archivo y plantear un mejor algoritmo para *process_lines* podrían ser algunos de los puntos claves para mejorar el proyecto en un futuro.