Image Reader

Progetto Mid-Term

Andrea Neri

Obiettivo del progetto

- Implementare un Image Reader in cui l'utente possa definire una cartella da cui prelevare le immagini facendo sì che il software si occupi del caricamento in modo non bloccante per renderle visibili.

- L'elaborato è stato sviluppato in Java e prevede un'implementazione sequenziale e una parallela, in modo da verificarne tempistiche di esecuzione e Speedup tra le due versioni.

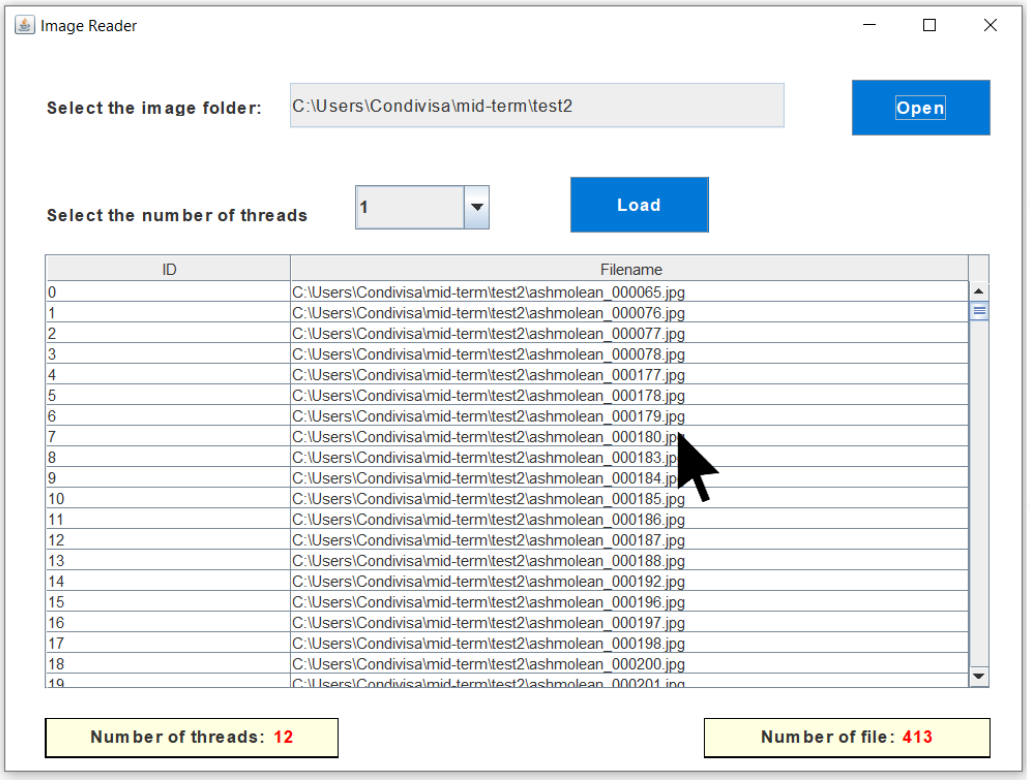
- L'utente può interagire con il programma tramite un'interfaccia grafica sviluppata in Swing.

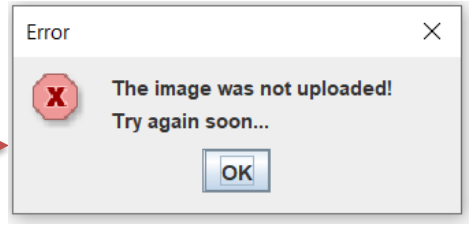
Descrizione del funzionamento

Dal punto di vista dell'utente

- Specifica la cartella in cui sono presenti le immagini
- Specifica il numero di core con cui vorrà eseguire il programma
- Al click del tasto load il software inizia a caricare le foto
- Nella tabella verranno visualizzati tutti i path assoluti delle foto
- Durante il caricamento l'utente può aprire le foto facendo doppio click su una riga della tabella
- Se la foto è già stata caricata in memoria principale la foto verrà visualizzata
    - Altrimenti riceverà un messaggio di errore, che lo invita a riprovare più tardi

Interfaccia grafica





Descrizione del funzionamento

Dal punto di vista del software

Il progetto sfrutta le seguenti classi per il funzionamento:

- **Image:** è l'oggetto che si occuperà di visualizzare l'immagine
- **Task:** si occupa ci caricare (e decomprimere) il jpg delle immagini contenute nel proprio subset all'interno del buffer condiviso e sincronizzato
- **Worker:** classe che implementa il paradigma fork-join, si occupa di creare, in base al numero di core selezionati, i subset di immagini, istanziare i Task, avviarli ed attendere la loro terminazione.
- **MainGUI:** classe che si occupa della creazione dell'interfaccia grafica, della creazione e distruzione del thread Worker e delle gestione dell'interfaccia.

Classe Task

La classe estende RecursiveTask che permette di implementare e definire un singolo Task, ovvero un compito all'interno del paradigma Fork-Join.
Ogni Task ha il compito di caricare un certo numero di immagini predefinito all'interno di un buffer condiviso tra tutti i task.

```
protected Boolean compute() {
    for (int i = startIndex; i < endIndex; i++){
        this.sharedBuffer.set(i, new Image(path[i].getPath()));
    }
    System.out.println("[Thread " + this.ID + "]: " +"finished");
    return true;
}
```

Classe Worker

Entrambe le versioni (parallela e sequenziale) sono implementate all'interno di questa classe:

- Se l'utente imposta un numero di **thread = 1**, viene eseguita la versione sequenziale
- Se l'utente imposta un numero di **thread > 1**, viene eseguita la versione parallela

Classe Worker

```
if (threadNumber == 1) {
     Task singleTask = new Task(0, images, sharedBuffer, 0, totalFile);
     singleTask.fork();
     singleTask.join();
     return;
```

```
int ID = 0;
ArrayList<ForkJoinTask<Boolean>> tasks = new ArrayList<>(threadNumber);
for (int i = 0; i < threadNumber; i++) {
     int startIndex = i * chunkDimension;
     int endIndex = (i + 1) * chunkDimension;
     if (i != threadNumber - 1) {
               tasks.add(new Task(ID, images, sharedBuffer, startIndex, endIndex));
               ID++;
     } else {
               tasks.add(new Task(ID, images, sharedBuffer, startIndex, endIndex +
remainingImage));
               ID++;
          }
}
```

Classe Worker

Divisione del lavoro

Nel caso di esecuzione parallela:

- I thread 0,…, thread N-2 caricheranno $\frac{Numero\ di\ immagini}{N}$ immagini
- Il thread N-1 caricherà $\frac{Numero\ di\ immagini}{N} + (numero\ di\ immagini)\ mod\ N$

Nel caso di esecuzione sequenziale:

- Il singolo thread andrà a caricare l'intero set di immagini

Versione con buffer privati

Per avere una metrica di paragone tra l'utilizzo di variabili condivise e private la prima implementazione è stata modificata in modo che ogni Task avesse il proprio buffer privato su cui caricare le immagini.

Modifiche effettuate:

- Le istanze della classe Task non utilizzeranno più un buffer di tipo synchronizedList, ma utilizzeranno un ArrayList privato, di conseguenza il metodo *compute* restituirà l'ArrayList riempito con le immagini appartenenti al proprio subset.

- La classe Worker alla creazione di un Task non fornirà più il buffer sul quale andare a caricare le foto, ma sarà il Task stesso a crearselo.

- Un'altra modifica viene applicata dopo la conclusione di tutti i Task, all'interno di un foreach tutti i buffer privati vengono riuniti in un singolo ArrayList.

```
for (ForkJoinTask<ArrayList<Image>> task :tasks) {
    ArrayList<Image> img = task.invoke();
    imageToLoad.addAll(img);
}
```

Macchina utilizzata per i test

HARDWARE:

- CPU: AMD Ryzen 2600, 2,4 GHz clock, 6 core/12 thread
- RAM: 16 GB DDR4 3200 MHz
- SSD NVME: Sabrent SB-ROCKET-512

SOFTWARE:

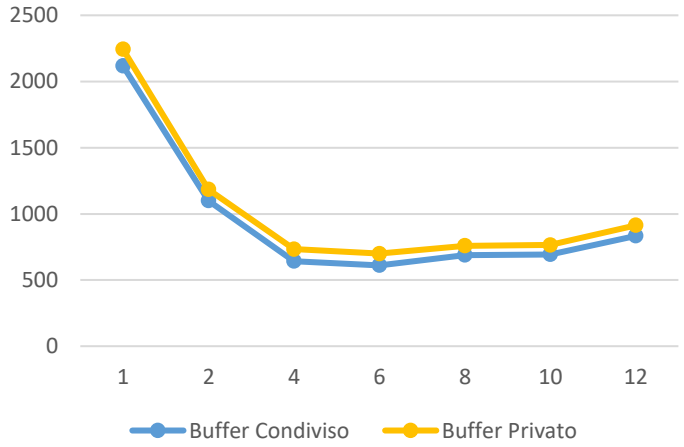
- JDK: 15.2.0
- IDE: Eclipse 4.19.0
- Windows Builder

NB: per avere dei dati meno influenzati dal sistema operativo e dallo scheduler, ogni singolo test è stato eseguito 5 volte, il valore riportato nelle tabelle è la media di tali valori.

Primo test

Set di 127 immagini

# Thread	Buffer Condiviso	Buffer Privato
1	2118	2243,4
2	1100,4	1185,2
4	643,6	733,2
6	611	700,4
8	689,4	759
10	693,6	765,4
12	834,6	913,4

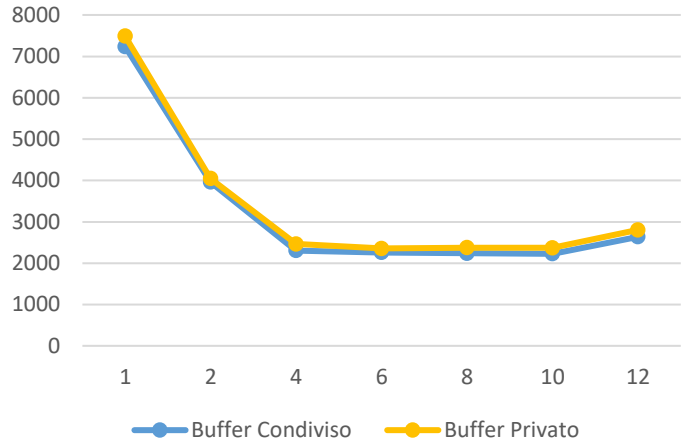


- Speedup (6th) buffer condiviso: 3,467
- Speedup (6th) buffer privato: 3,203

Secondo test

Set di 413 immagini

# Thread	Buffer Condiviso	Buffer Privato
1	7237	7488,4
2	3963,2	4045,6
4	2305,2	2465,8
6	2258,2	2355
8	2239,6	2375,4
10	2229,2	2370,4
12	2639,6	2804



- Speedup (10th) buffer condiviso: 3,246
- Speedup (10th) buffer privato: 3,159

Conclusione

Aumentando la dimensione del set di immagini da caricare, si può notare come è possibile aumentare il numero di thread da utilizzare per continuare ad avere un miglioramento nei tempi di esecuzione.

In entrambi i test, l'uso di un buffer privato per ogni singolo Task al posto di un buffer condiviso e sincronizzato tra tutti i thread, non ha portato dei vantaggi a livello di tempistiche.

L'implementazione con buffer privato è stata fatta solo per avere una metrica di paragone tra l'utilizzo di variabili condivise e private.