UNIVERSITÀ
DEGLI STUDI
FIRENZE

ImageReader

Alessandro Mini

UNIVERSITÀ DEGLI STUDI FIRENZE

Indice

UNIVERSITÀ DEGLI STUDI FIRENZE

Introduzione

Obiettivo: Leggere in modo sequenziale e parallelo *non bloccante* delle immagini da una cartella.

- L'utente specifica una folder contente immagini jpg.
- Il programma inizia a caricare le immagini.
- Mentre il programma carica le immagini l'utente può visualizzarle tramite doppio click sulla tabella, se un immagine non è ancora stata caricata, restituirà un errore e chiederà di riprovare.

Si è scelto di utilizzare **Java 14** con interfaccia **Swing**.

UNIVERSITÀ DEGLI STUDI FIRENZE

Screenshot

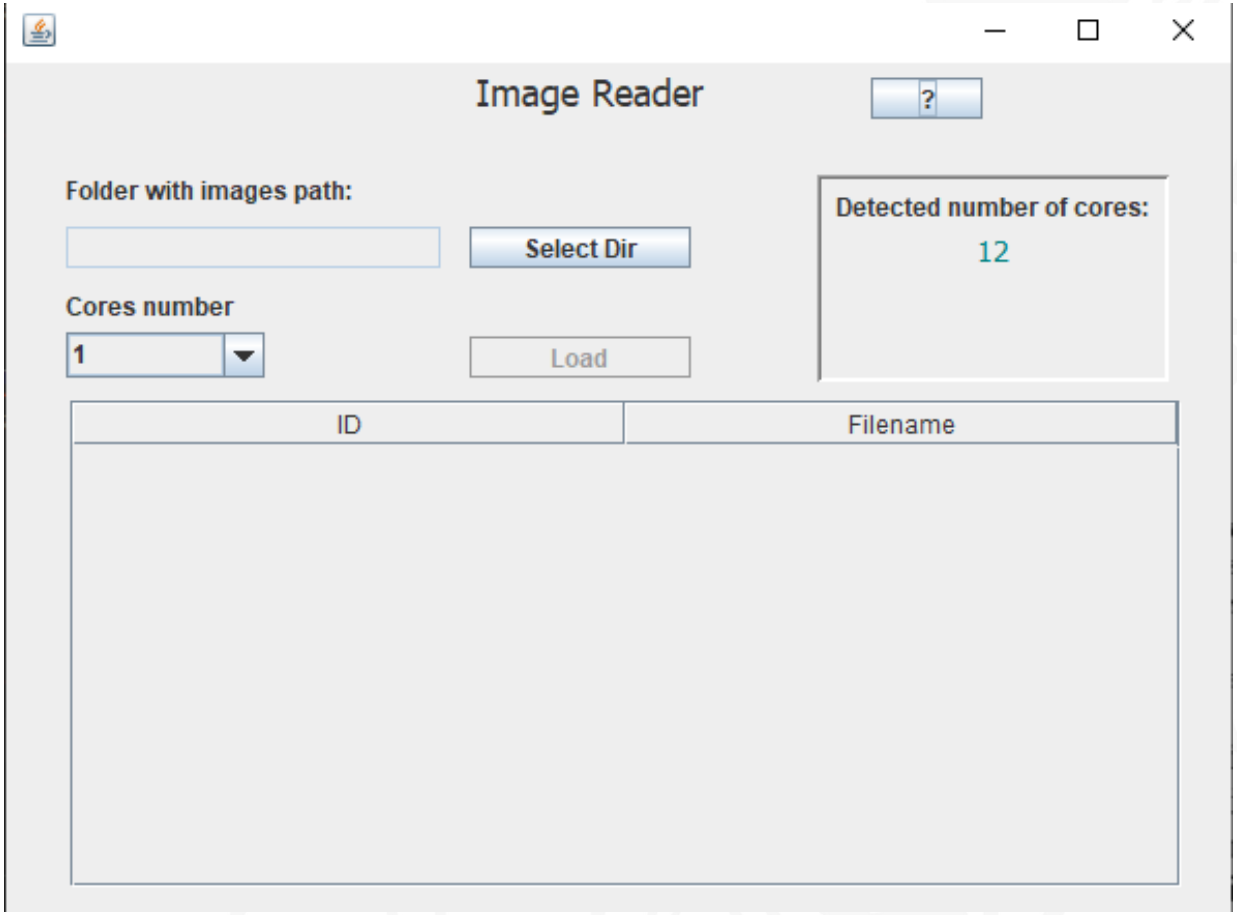

Figure 1: Interfaccia grafica del programma.

Screenshot

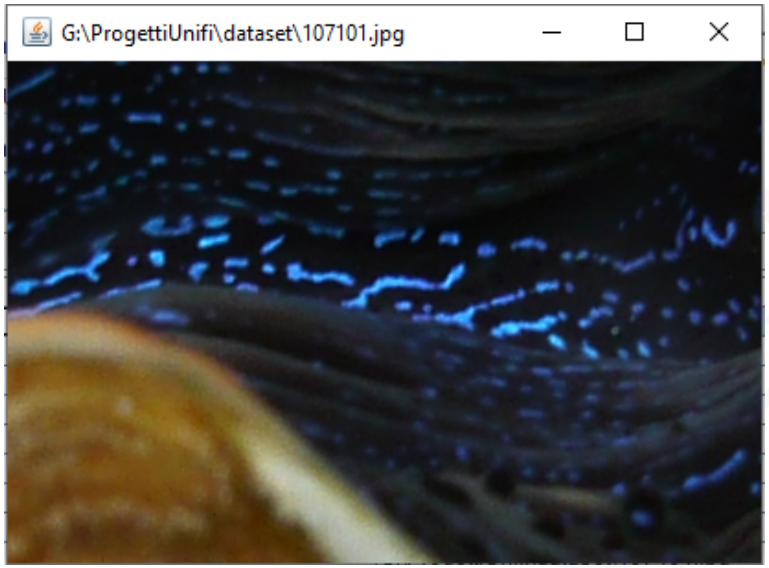

Figure 2: Immagine selezionata dall'utente che è stata caricata

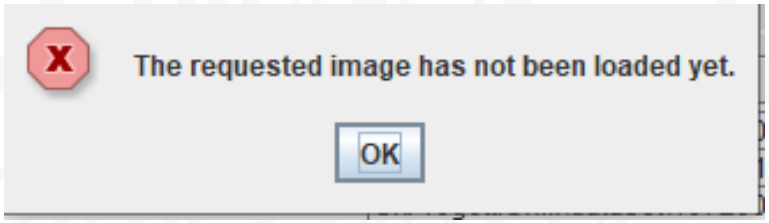

Figure 3: Se l'immagine non è ancora stata caricata restituisce un errore

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione del funzionamento

La versione sequenziale e parallela del programma condividono lo stesso codice, il programma permette di scegliere il numero di core.

- Se $cores = 1$ allora si eseguirà la versione sequenziale.
- Se $cores > 1$ si eseguirà la versione parallela.

UNIVERSITÀ DEGLI STUDI FIRENZE

Descrizione del funzionamento

Si definiscono i seguenti oggetti:

- Image: rappresenta un immagine, il costruttore carica direttamente l'immagine fisica (dal S.O) in memoria.
- ParallelWorker: classe che implementa il parallelismo attraverso un paradigma di tipo **fork-join**.
- LoadTask: classe che rappresenta un task di tipo "Caricamento di immagini", un task caricherà porzioni diverse della cartella sorgente.
- Main: entry point del programma, istanza gli oggetti e la GUI.

UNIVERSITÀ DEGLI STUDI FIRENZE

Codice di ParallelWorker:

```
if (numCore == 1) {
    loadTask sequentialTask = new loadTask(0, immagini, images, 0, Utils.countFiles(dir));
    //Creo un unico task che ricopre tutte le immagini
    ...
    return;
}
//Se ho più di un core divido il lavoro su più task.
...
for (int i = 0; i < numCore; i++) {
    // For core 1...N-1 i assign a chunk.
    int start = i * chunk;
    int end = (i + 1) * chunk;
    if (i == numCore - 1) {
        C.add(new loadTask(threadID, immagini, images, start, end + resto));
        threadID++;
    }
    else {
        //if N-1 then work + reminder.
        C.add(new loadTask(threadID, immagini, images, start, end));
        threadID++;
    }
}
```

Codice di LoadTask:

```
protected Boolean compute() {
        for (int k = startIndex; k < endIndex; k++) {
                this.immagini.set(k, new Image(baseFolder[k].getAbsolutePath()));
        }
        return true;
}
```

Immagini è una struttura dati sincronizzata che contiene oggetti di tipo Image.

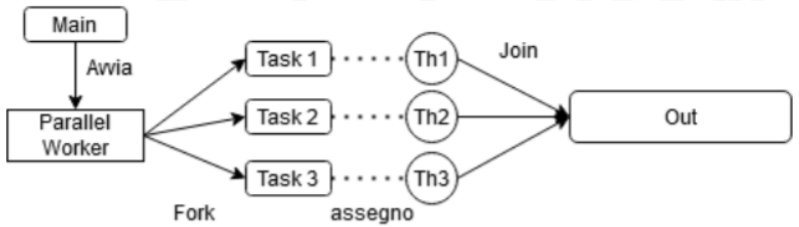

Figure 4: Rappresentazione del paradigma utilizzato.

Dettaglio sullo scheduling del lavoro

Il lavoro viene suddiviso nel seguente modo:
Se si decide di eseguire il programma in parallelo:

- I core $c_0, c_1, c_{numCore-1}$ eseguiranno $\frac{N}{C}$ caricamenti.
- Il core $c_{numCore-1}$ eseguirà $\frac{N}{C}$ caricamenti e la parte restante dalla divisione, l'ultimo core è quello che eseguirà lavoro maggiore.

Nel caso di esecuzione sequenziale:

- Il core c_0 eseguirà il caricamento di *tutte* le immagini, in modo sequenziale.

Considerazioni

Si può notare che:

- La scelta del dataset influsce molto sulle prestazioni.
- Le immagini devono essere in una quantità e dimensione tale da non superare quella dell' heap della JVM.
- Il sistema operativo ha un ruolo importante nello speedup, il programma eseguirà molte richieste di accesso, il S.O dovrà schedularle.

Si considera la seguente macchina di test:

CPU	Ryzen 5 3600 (6 core)
RAM	16 GB DDR4
HDD	WD Blue 1TB
M2	Sabrent Q4

Figure 5: Macchina di test

Test (1)

Sul seguente dataset ”lear.inrialpes.fr/people/jegou/data.php si” ha un tempo sequenziale di 18682*ms* e parallelo:

Cores utilizzati:	Tempo HDD	Speedup	Efficienza
2	10737	1,74	0,87
4	8037	2,32	0,58
6	7331	2,55	0,42

Figure 6: Risultati sul primo dataset

Test (2)

Su un secondo dataset di 900 immagini con risoluzione 400 x 600 si ha un tempo sequenziale di 5686*ms* e parallelo:

Cores:	Tempo HDD	Speedup	Efficienza
2	2982	1,91	0,95
4	1818	3,13	0,78
6	1672	3,40	0,57

Figure 7: Risultati sul secondo dataset

Ripetendo il secondo test su M2 si ha:

Cores:	Tempo M2	Speedup	Efficienza
2	2979	1,88	0,94
4	1712	3,27	0,82
6	1685	3,32	0,55

Figure 8: secondo dataset ed M2

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusioni

Si è costruita una versione parallela e sequenziale del programma, si ottiene sempre uno *speedup* sebbene questo sia fortemente influenzato dalla macchina e dal *dataset*.
Nel calcolo dello speedup reale va tenuto conto di un forte intervento del sistema operativo e dell' hardware.