Ricerca parallela dei Bigrammi in Java

Marco Calamai, *Elia Mercatanti*

Università degli Studi di Firenze
Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Anno Accademico 2019-2020

Scopo del progetto

- Implementazione sequenziale e parallela in Java dell'algoritmo che calcola le frequenze di bigrammi in un insieme di file di testo.
  - Bigrammi di lettere.
  - Bigrammi di parole.
- Confronto dei tempi di esecuzione e speedup ottenuti.

Cos'è un n-gramma

- Un n-gramma è una sequenza contigua di n elementi in un testo.
- Gli elementi possono essere parole, lettere, sillabe etc..
- Le modalità di selezione dei caratteri o delle parole dal testo variano molto in base all'obiettivo dell'applicazione.

Nel nostro progetto:

- Sono state prese in considerazione bigrammi di lettere e parole adiacenti da un insieme di libri testuali.
- Considerando esclusivamente le lettere dell'alfabeto inglese, niente punteggiatura o spazi.

Implementazione Sequenziale - Recupero Testi

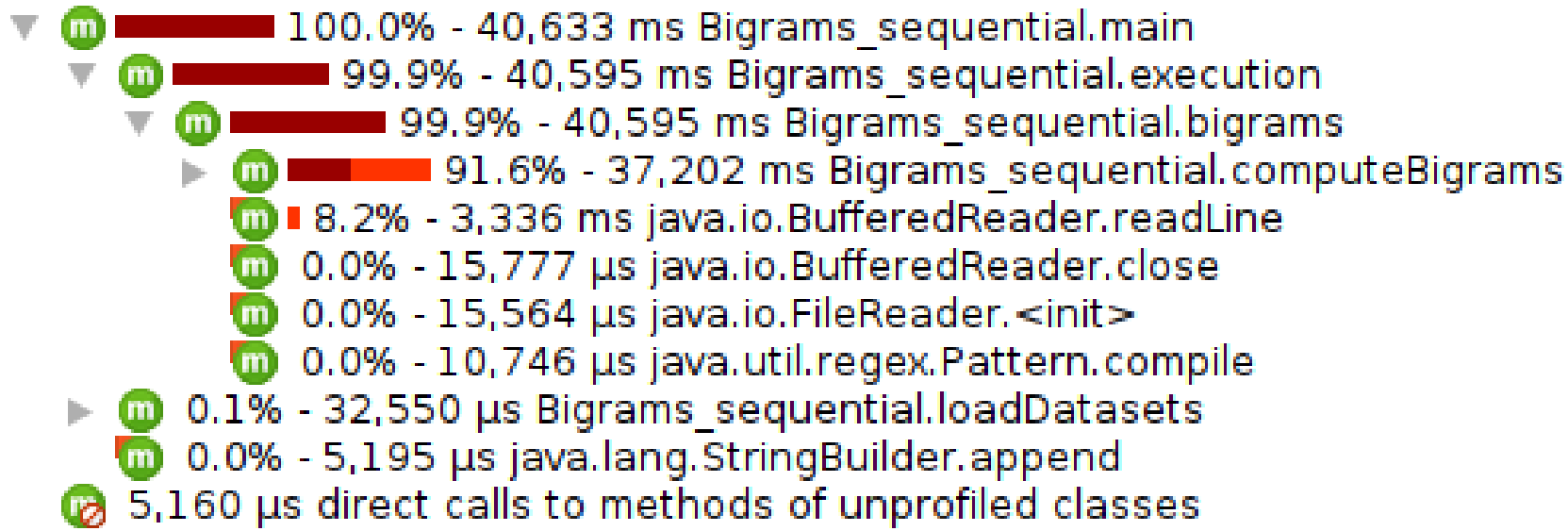
```
try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get("texts"))){
    for (Path path : stream) {
        if (!Files.isDirectory(path)) {
            txtListMain.add(path.getFileName().toString());
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println("Input txt files: " + txtListMain + "\n");
```

Implementazione Sequenziale - Calcolo Bigrammi

```
while ((line = buffer.readLine()) != null) {
    m = searchToken.matcher(line);
    if(!m.find()) {
        continue;
    }
    m = searchToken.matcher(line);

    while(m.find()) {
        nextToken = m.group();
        nGram = (prevToken +  nextToken);
        dict.merge(nGram, 1, (prev, one) -> prev + one);
        prevToken = nextToken;
    }
}
```

Analisi versione Sequenziale con Profiler Java



Versioni Parallele - Schema Utilizzato

Schema implementato:

- Produttore/Consumatore.
  - Un solo produttore che legge i files sequenzialmente.
  - Più consumatori in parallelo che calcolano i bigrammi sulle stringhe generate dal produttore.

Per gestire la comunicazione:

- Una coda concorrente non bloccante nella quale il produttore inserisce le stringhe da elaborare che i consumatori estraggono.
- Un hash map concorrente, nel quale i consumatori inseriranno il numero delle occorrenze dei bigrammi trovati.

I consumatori sono stati gestiti con un thread pools.

Implementazione parallela - Produttore

```
while ((this.line = buffer.readLine()) != null) {
    this.m = validTxtLine.matcher(line);
    if(!this.m.find()) {
        continue;
    }

    this.lastLine = this.line;
    this.mergedLine = this.line;
    for (int i = 0; i < 20; i++) {
        if ((this.line = buffer.readLine()) == null) {
            break;
        }

        this.m = validTxtLine.matcher(this.line);
        if(!this.m.find()) {
            continue;
        }

        this.lastLine = this.line;
        this.mergedLine = this.mergedLine + " " + this.line;
    }
    notBlockingqueue.add(this.lastToken + " " + this.mergedLine);
    this.m = lastTokenPattern.matcher(this.lastLine);

    if(this.m.find()) {
        this.lastToken = this.m.group();
    }
    else {
        this.lastToken = "";
    }
}
```

Implementazione parallela - Consumatore

```
while (true) {
    this.textLine = notBlockingqueue.poll();
    if (this.textLine == null) {
        continue;
    }
    if (this.textLine == NO_MORE_MESSAGES) {
        break;
    }

    this.m = searchToken.matcher(this.textLine);
    if(this.m.find()) {
        this.prevToken = this.m.group();
    }
    while(this.m.find()) {
        this.nextToken = this.m.group();
        this.nGram = (this.prevToken + this.nextToken);
        dict.merge(this.nGram, 1, (prev, one) -> prev + one);
        this.prevToken = this.nextToken;
    }

}
```

Implementazione parallela - Fase di Merge Consumatore

```
for (String name : this.dict.keySet()) {
    String key = name;
    Integer value = this.dict.get(name);
    globalDict.merge(key, value, (prev, update) -> prev + update);
}
```

Considerazioni

- La fase di lettura del file del produttore risulta sequenziale in quanto lettura da memoria di massa non parallelizzabile.
- Sono stati scelti 1 thread produttore e 4 threads consumatori.
- 4 consumatori in quanto test fatti su una CPU a 4 core.
- Altri test eseguiti variando il numero di threads consumatori e produttori.
  - Aumentare il numero di threads produttori non porta ad alcun beneficio, peggiora il tempo di esecuzione complessivo.
  - Sono stati utilizzati anche più di 4 consumatori ma non ha portato rilevanti benefici.

Considerazioni

- Utilizzo di una coda non bloccante, in quanto ci si aspetta una contesa media e basso numero di threads totali.
- Test utilizzando una coda bloccante, risultati paragonabili nelle stesse condizioni.
- Il produttore concatena in una stringa più righe del file per creare più lavoro per i consumatori.
- Il numero di stringhe che il produttore cerca di concatenare è stato scelto pari a 20. Compromesso tra:
  - Un numero troppo basso che rallenterebbe il tempo di esecuzione totale del produttore.
  - Un numero troppo alto che porterebbe a troppe operazioni di concatenazione da parte del produttore.

Test

- Eseguiti su CPU quad core i7 3770K.
- Testi presi dal sito web `https://www.gutenberg.org/`.
- Sono stati scelti 15 libri.
- Test di grandi dimensioni su file copiati più volte, datasets di dimensioni complessive pari a 105, 315 e 630 e 1260 testi.
- Per calcolare i tempi di esecuzione è stato eseguito l'algoritmo 5 volte e fatta una media dei tempi risultanti.

BIGRAMS

# FILES		Letters	Words
15	**Sequenziale**	1.86	1.55
	Parallelo	0.83	1.15
	Speedup	2.24	1.35
105	**Sequenziale**	12.64	9.5
	Parallelo	5.1	5.42
	Speedup	2.48	1.75
315	**Sequenziale**	39.17	28.5
	Parallelo	15.09	14.8
	Speedup	2.60	1.93
630	**Sequenziale**	77.15	53
	Parallelo	31.11	24.74
	Speedup	2.48	2.14
1260	**Sequenziale**	150.29	117.11
	Parallelo	55.48	52.35
	Speedup	2.71	2.24

Versione Parallela Alternativa

- Obiettivo: ridurre il tempo di esecuzione del produttore.
- Modifica: produttore legge e inserisce ogni libro di testo in un'unica stringa.
- Risultati:

BIGRAMS

# FILES		Letters	Words
15	**Sequenziale**	1.61	1.34
	Parallelo	0.76	0.963
	Speedup	2.12	1.39
105	**Sequenziale**	11.4	9.12
	Parallelo	4.19	4.74
	Speedup	2.72	1.92
315	**Sequenziale**	34.19	27.17
	Parallelo	11.76	12.56
	Speedup	2.91	2.16
630	**Sequenziale**	67.41	49.94
	Parallelo	24.33	21.36
	Speedup	2.77	2.34