Q-grams in Java threads

Esame di PC

1°Parte

Cos'è un q-gramma

- Un q-gramma è una sequenza di q-items dato un testo
- Gli items possono essere lettere, parole, sillabe o fonemi.
- In informatica i q-grammi sono regolarmente impiegati in criptografia.
- In questo progetto ci si focalizzerà solo sui bigrammi.

Dataset

- Per testare i programmi è stato costruito un dataset composto da 9 libri presi dal sito "Progetto Gutenberg". Ogni libro pesa all'incirca 360Kb e da questi 9 libri si sono costruiti i seguenti dataset ripetendoli
  - 101 (35.5Mb)
  - 51 (17.9Mb)
  - 41 (14.4 Mb)
  - 31 (10.9Mb)
  - 21 (7.38Mb)
  - 11 (3.87Mb)

Implementazioni

- Sono state sviluppate due implementazioni
  - Sequenziale
  - Parallela

Implementazione sequenziale

- La versione sequenziale è un programma Java che si compone di due classi:
  - BookDataset
  - Qgramma

BookDataset

- è una classe che ha il compito di recuperare i libri salvati in una cartella interna al progetto.
- Una volta che questi libri sono stati recuperati, vengono resi disponibili come una lista di stringhe tramite il metodo getDataset()

```
public List<String> getDataset() throws IOException {
    String path = "/home/matteo/IdeaProjects/Qgram_PC/datasetBook";
    for (int i = 0; i < numBook; i++) {
        FileReader fileBook;
        fileBook = new FileReader( s: path+"/book_" + i + ".txt");

        BufferedReader readerBook;
        readerBook = new BufferedReader(fileBook);

        StringBuilder book = new StringBuilder();
        String line;
        while (true) {
            line = readerBook.readLine();
            if (line == null)
                break;
            book.append(line+"\n");
        }
        dataset.add(book.toString());
    }

    return dataset;
}
```

Qgram

- La classe Qgram ha il compito di analizzare i q-grammi di un testo.
- Il metodo qgrams() accetta in input una stringa con lettere in minuscolo di lunghezza n e itera su di essa usando una finestra scorrevole di dimensione fissa q.
- Se la finestra trova un q-gramma allora, si procede a inserirlo nella hashmap e poi avanza verso destra.
- Se la finestra scorrevole trova un sequenza che contiene spazi o caratteri speciali, allora avanza verso destra.

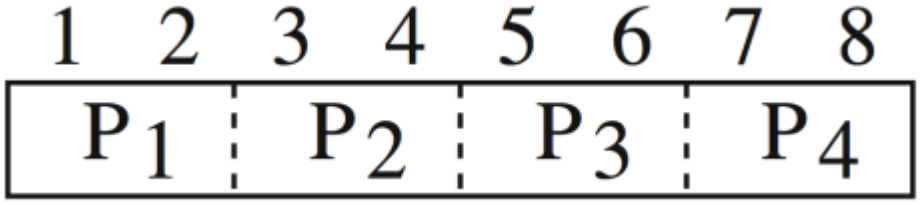
```java
public HashMap<String, Integer> qGrams(String text) {
    //text = text.toLowerCase();
    String tupla;
    for (int i = 0; i < text.length() - q + 1; i++) {
        tupla = text.substring(i, i1: i + q);
        if (onlyChar(tupla)) {
            if (grams.containsKey(tupla)) {
                grams.replace(tupla, v: grams.get(tupla) + 1);
            } else {
                grams.put(tupla, v: 1);
            }
        }
    }
    return grams;
}

private boolean onlyChar(String tuple) {
    int i = 0;
    while (i < q) {
        int symNum = (int) tuple.charAt(i);
        if (symNum == 224 || symNum == 232 || symNum == 233 || symNum == 236 || symNum == 242)
            i++;
        else if (symNum >= 97 && symNum <= 122)
            i++;
        else
            return false;
    }
    return true;
}
```

Implementazione Parallela

- Da java 8 in poi sono state introdotte molte novità per scrivere i thread: come i Callable che restituiscono un risultato a differenza dei Runnable.
- Usando questi nuovi strumenti, possiamo riscrivere la classe che analizza i q-grammi nel seguente modo:

ParallelQGrams

- La nuova classe si chiama ParallelQGrams e ha tutti gli stessi metodi della classe Qgrams.
- qGrams, adesso si chiama call() e non prende in input nessuna stringa, questo perchè si deve rispettare l'interfaccia Callable.
- Per la versione parallela è stato deciso di applicare la tecnica della data decomposition per distribuire il lavoro ai threads.



```
private void divideText(String text) {
    double textLen = text.length();
    int i, j;
    double work = Math.ceil(textLen / processors);
    i = 0;
    j = (int) work;
    String subText = null;
    while (j < textLen) {
        while (text.charAt(j) != ' ') {
            j++;
        }
        subText = text.substring(i, j);
        i = j;
        j = (int) (i + work);
        taskQGrams.add(new ParallelQgrams(subText, q));
    }
    subText = text.substring(i, (int) textLen);
    taskQGrams.add(new ParallelQgrams(subText, q));
}
```

```
public HashMap<String, Integer> call() throws Exception {
    String tuple;
    for (int i = 0; i < text.length() - q + 1; i++) {
        tuple = text.substring(i,  i1: i + q);
        if (onlyLetters(tuple)) {
            if (qGrams.containsKey(tuple)) {
                qGrams.replace(tuple,  v: qGrams.get(tuple) + 1);
            } else {
                qGrams.put(tuple,  v: 1);
            }
        }
    }

    return qGrams;
}
```

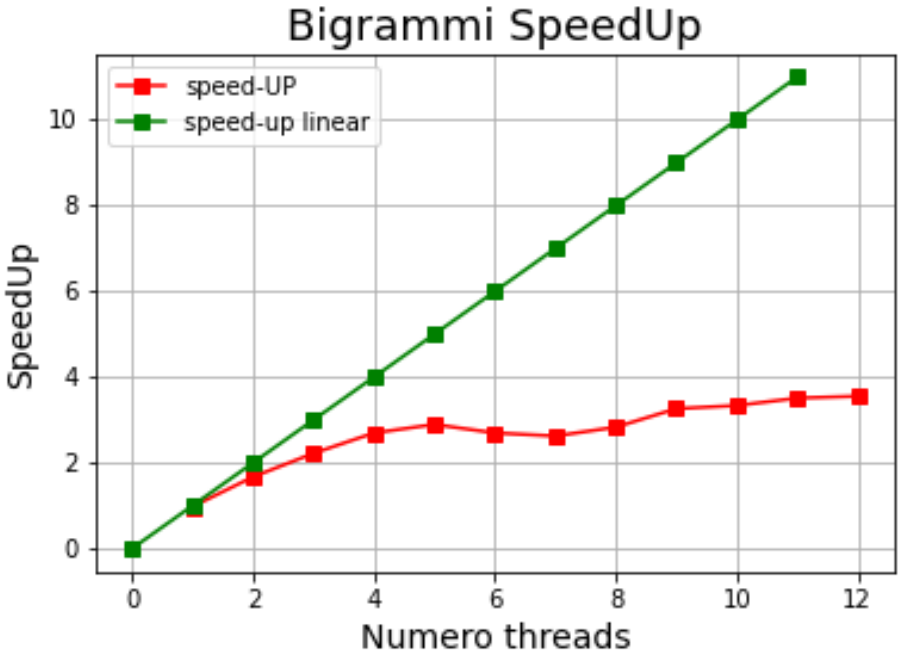
```
public Map<String, Integer> qGrams(String text)
        throws InterruptedException, ExecutionException {
    divideText(text);
    List<Future<HashMap<String, Integer>>> resultsQgrams = exec.invokeAll(taskQGrams);

    Map<String, Integer> mergeAll = new ConcurrentHashMap<String, Integer>();
    for (Future<HashMap<String, Integer>> result : resultsQgrams) {
        result.get().forEach((k, v) -> mergeAll.merge(k, v, Integer::sum));
    }
    taskQGrams.clear();
    return mergeAll;
}
```

Analisi delle performance

- Per condurre le analisi delle prestazioni è stata usata una macchina MSI con le seguenti specifiche hardware:
  - Processore: Intel® Core™ i7-10750H CPU @ 2.60GHz 6 core
  - Ram: 16GB DDR4 3200Mhz
  - Hardisk: SSD 1TB

Analisi delle performance

- L'analisi è stata condotta eseguendo sulle diverse versioni dei dataset.
- E' stato fissato q=2 per ricavare dei bigrammi dai testi.



Speedup

- Con 12 thread il programma è 3 volte più veloce della versione sequenziale.
- Lo speedup inizia a divergere dallo speedup lineare già a partite da 3 thread e, man mano che si aumentano i thread, tale divergenza diventa sempre più netta.
- Osservando lo speedup ottenuto, si ritiene che il programma parallelo soffra del fenomeno del false sharing.