

Bigram and Trigram Computing

Paolo Le Piane
Manuel Natale Sapia



Obiettivo

- Implementazione di un generatore di bigrammi e trigrammi di lettere estratti da un testo selezionato

Svolgimento

- Tre versioni di implementazione:
 - versione sequenziale in Java
 - versione parallela in Java
 - versione parallela in C++ con OpenMP
- Risultati e analisi:
 - analisi speed up
 - analisi tempi di esecuzione

Dataset:

- Testo del romanzo “Orgoglio e Pregiudizio” modificato per ottenere dimensioni diverse utili per l’analisi, in particolare: 75 Kb, 150 Kb, 250 Kb, 500 Kb, 1 Mb, 2 Mb, 4 Mb, 8 Mb, 16 Mb, 32 Mb, 64 Mb.

Specifiche PC utilizzato:

- Processore Intel Core i7-6700HQ da 2.60GHz con turbo boost fino a 3.5GHz
- GPU NVIDIA GeForce GTX 970M con 1280 core e 3GB di memoria dedicata
- 16GB di memoria RAM

Versione Sequenziale in Java

Funzioni implementate:

- myReadFile()
- generateNgrams()
- frequency()
- myWriteFile()
- main()

- *generateNgrams()*
percorre la stringa di
testo e crea gli n-grammi
in base all'n scelto.

```
public static List<String> generateNgrams(int n, String str) {  
    String letters = "abcdefghijklmnopqrstuvwxyz";  
    boolean skip;  
    List<String> ngrams = new ArrayList<>();  
    for (int i = 0; i < str.length() - n + 1; i++) {  
        skip = false;  
        for (int k = 0; k < n ; k++) {  
            if (letters.indexOf(str.charAt(i+k)) == -1) {  
                skip = true;  
                break;  
            }  
        }  
        if (skip) continue;  
        ngrams.add(str.substring(i, i + n));  
    }  
    return ngrams;  
}
```

Versione Parallela in Java

- Implementazione in Java utilizzando l'interfaccia Callable.
- Funzione *create_chunks()* per ottenere gli intervalli per partizionare in blocchi la stringa con il testo.

```
public static List<Integer> create_chunks(int numb_th,
String str) {
    List<Integer> result = new ArrayList<>();
    int pos;
    result.add(0);
    int chunk = str.length() / numb_th;
    for (int k = 1; k < numb_th; k++ ) {
        pos = chunk * k;
        while (str.charAt(pos) != ' ') {
            pos += 1;
        }
        result.add(pos);
    }
    result.add(str.length());
    return result;
}
```

- Creazione di un `ThreadPoolExecutor` che genera e gestisce un `ThreadPool`.
- `call()` esegue il task asincrono e genera gli n-grammi relativi all'intervallo del testo selezionato.

```
public static void main(String[] Args) throws  
InterruptedException {  
    ...  
    ThreadPoolExecutor executor = (ThreadPoolExecutor)  
Executors.newFixedThreadPool(4);  
    List<Future<List<String>>> resultList = new  
ArrayList<>();  
    Instant start_t = Instant.now();  
    for (int i = 0; i < number_th; i++) {  
        CallMain prova = new CallMain(2, book, ck.get(i),  
ck.get(i+1));  
        Future<List<String>> result = executor.submit(prova);  
        resultList.add(result);  
    }  
    for (Future<List<String>> future : resultList) {  
        try  
        {  
            count = count + future.get().size();  
        }  
        ...  
    }  
}
```

Versione Parallela in C++ OpenMP

- Modello fork&join in OpenMP
- Metodi *myReadFile()* e *myWriteFile()* per la lettura da file e la scrittura su file.

```
regex letter("[^[:alpha:]]");
//apertura file
while (getline(inputFile, line)) {
    line = regex_replace(line, letter, " ");
    stringstream line2(line)
    while (getline(line2, word, ' ')) {
        //rendi word minuscolo e inseriscilo nel vettore
        di stringhe
    }
}
```




- Metodo per la creazione dei thread
generateNgramsParallel()
dove si usano i costrutti di *OpenMP*
- Vettore *ngrams* multidimensionale.
- Variabile *thread_index* fondamentale per il metodo *compute()*
- Per ogni parola genera $m - n + 1$ n-grammi.
- Metodo *frequency()*.

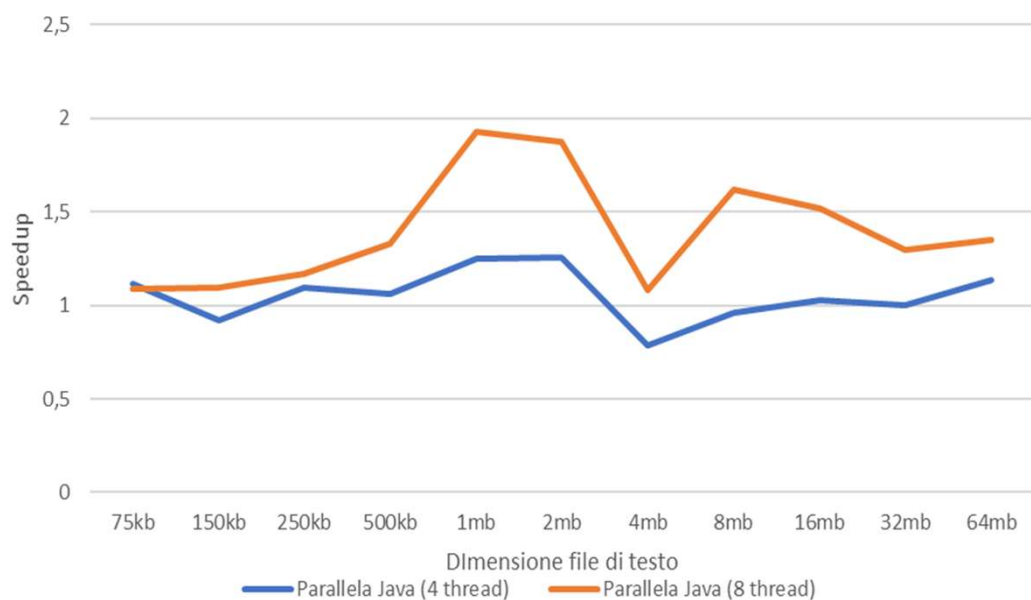
```
#pragma omp parallel private(thread_index,word)
num_threads(num_threads)
{
    thread_index = omp_get_thread_num();
    #pragma omp for
    for (int j = 0; j < strList.size(); j++) {
        word = strList[j];
        compute(thread_index, word, n, ngrams);
    }
}
```

```
if(!word.empty())
    limit = word.length() - n + 1;
    for(int i = 0; i < limit; i++){
        ngrams[thread_index].push_back(word.sub
str(i,n));
    }
```

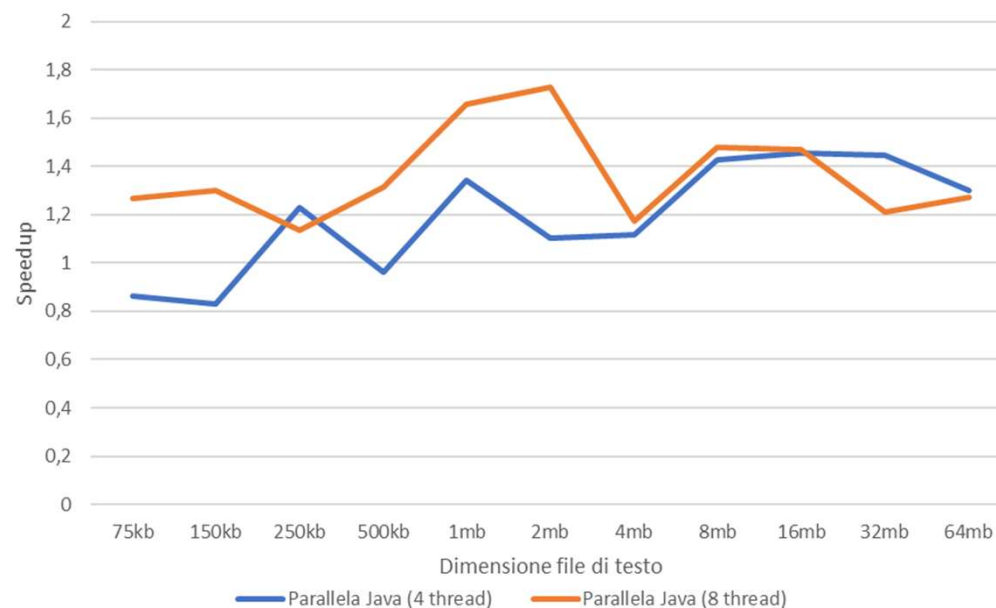


Confronto Speedup in Java

Confronto Speedup Bigrammi Java



Confronto Speedup Trigrammi Java

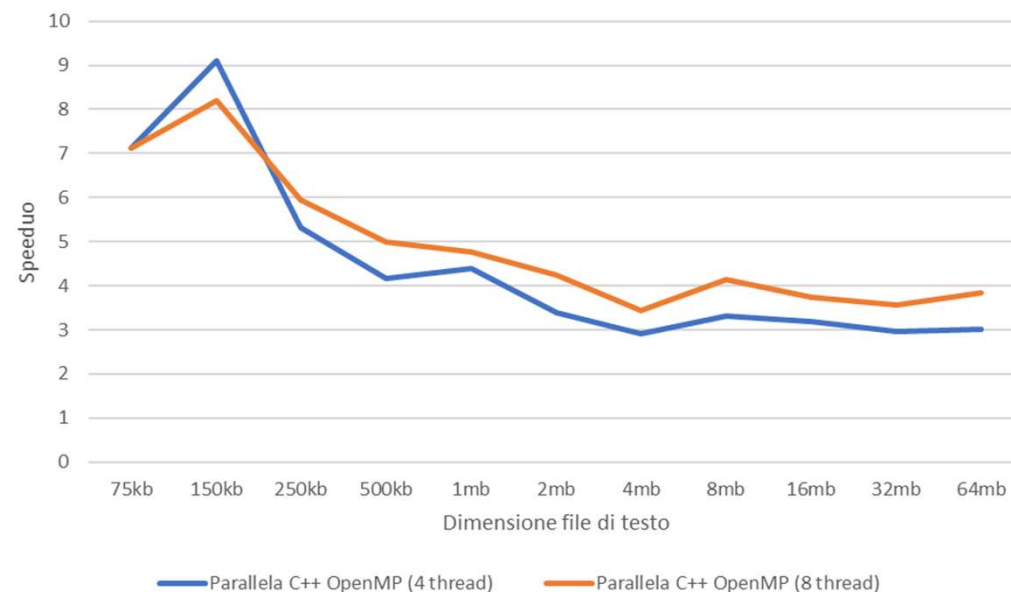


Confronto Speedup in C++ (OpenMP)

Confronto Speedup Bigrammi C++ (OpenMP)

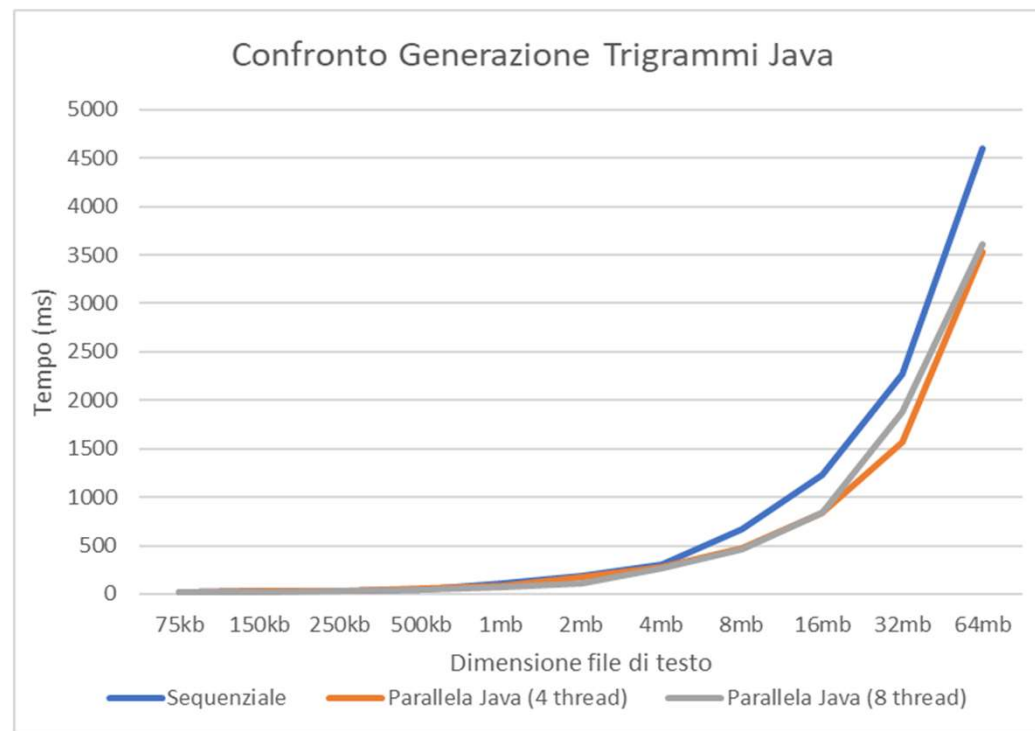
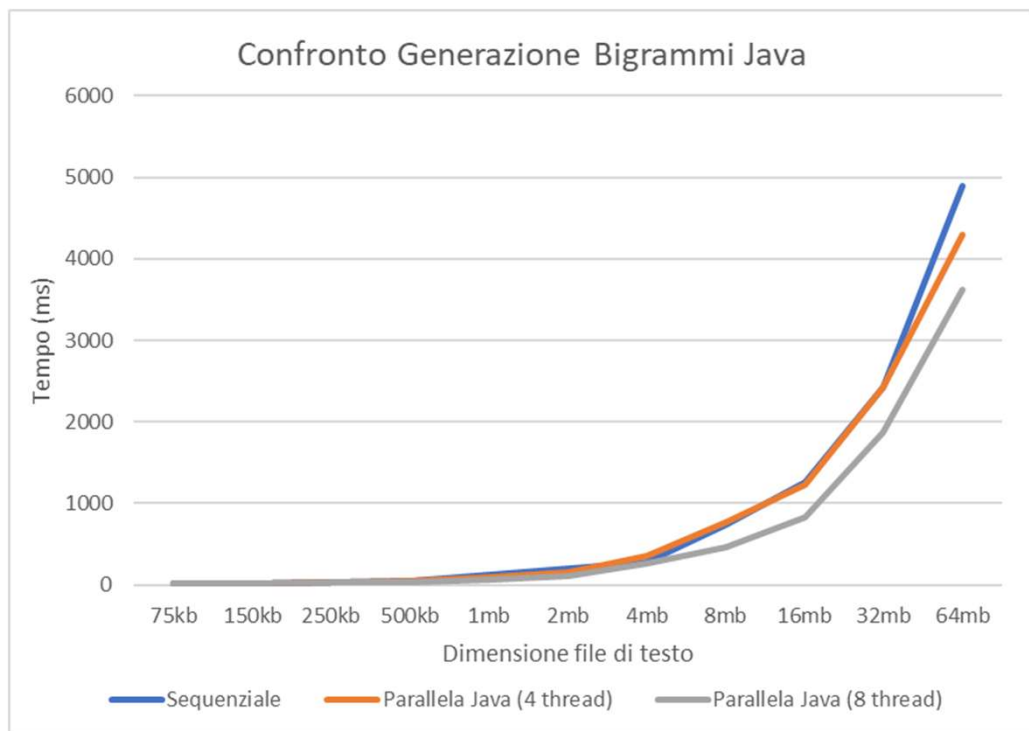


Confronto Speedup Trigrammi C++ (OpenMP)



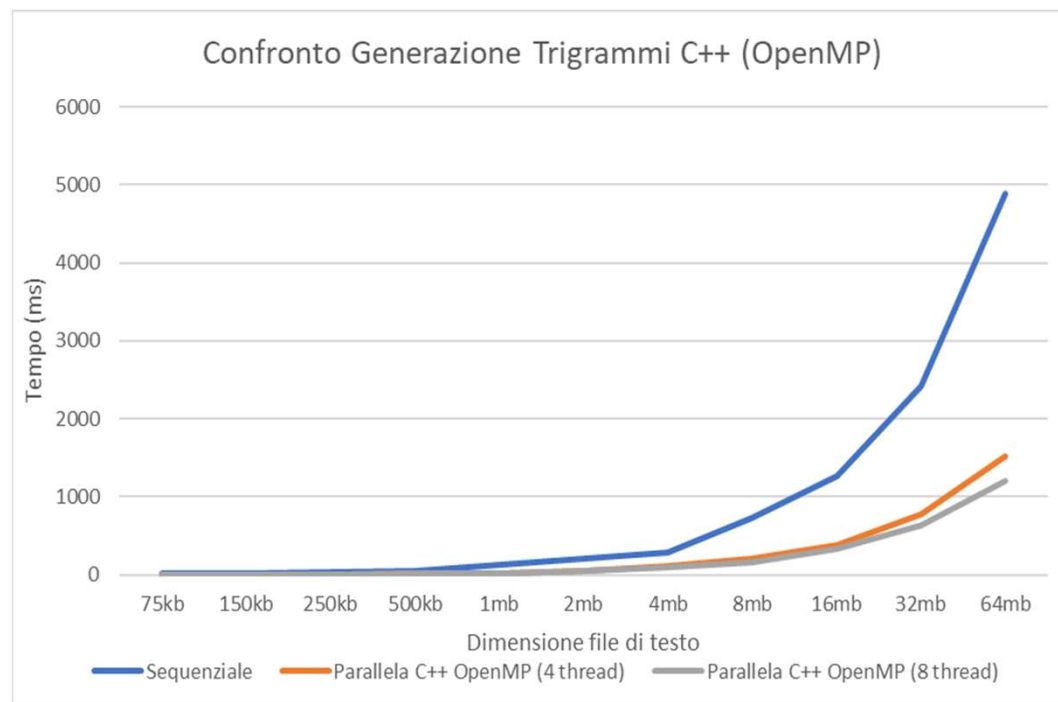
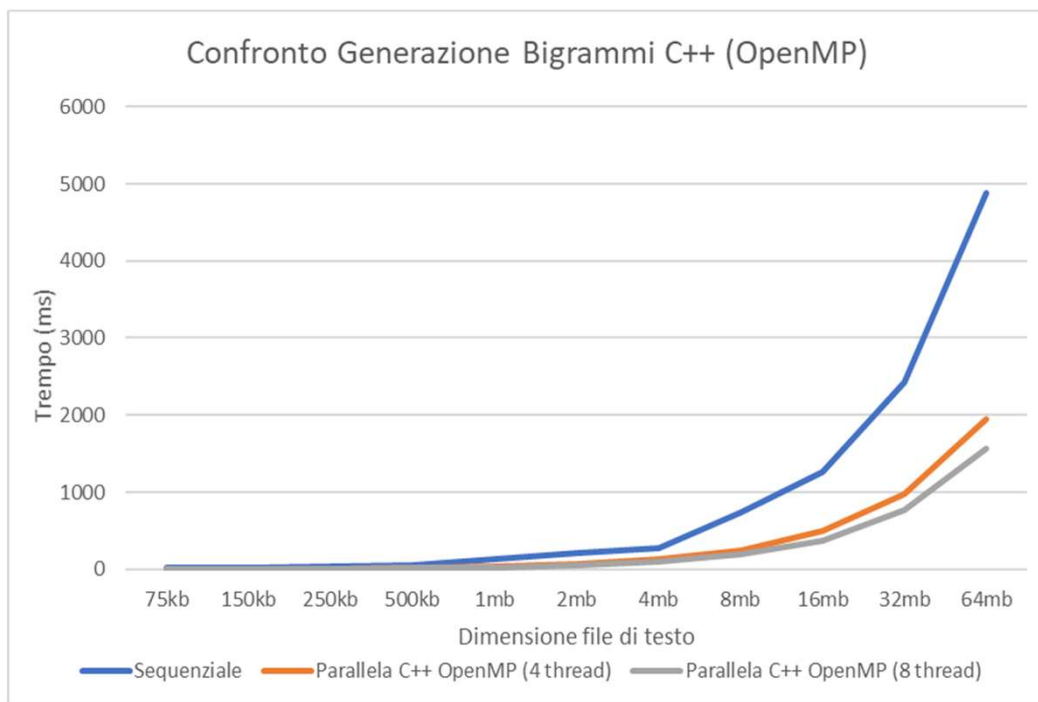


Confronto tempi di esecuzione in Java





Confronto tempi di esecuzione in C++ (OpenMP)



Conclusione

In questo progetto abbiamo:

- Implementato un framework per la generazione di bigrammi e trigrammi
- Sviluppato in una versione sequenziale (Java) e due parallele (Java e C++ OpenMP)
- Effettuato analisi sullo speed up e sulle tempistiche di esecuzione



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze
Parallel Computing

***Grazie
dell'attenzione***