Istogramma N-Grams OMP

Alessandro Marinai, Eleonora Ristori

Aprile 2023

1 Introduzione

Lo scopo dell'elaborato è la realizzazione di un programma che calcoli l'istogramma degli n-gram presenti in un testo. Sono state realizzate due versioni del codice per l'esecuzione del task:

- una versione sequenziale;
- una versione parallela che utilizza la libreria openMP.

Si riporta quindi una analisi delle prestazioni in termini di tempo di esecuzione e speed-up delle due versioni.

2 Struttura del codice

Il task del programma è realizzato da una classe denominata nGramCounter. Essa è caratterizzata da 2 attributi, ovvero la lunghezza degli n-gram, inizializzata nel costruttore della classe ed una mappa che associa ad ogni n-gram il numero di occorrenze nel testo. Sono inoltre definiti 3 metodi:

- il metodo privato extractNGramsFromWord() che restituisce l'array degli n-gram presenti in un vocabolo;
- il metodo countNGrams() che calcola l'istogramma degli n-grams in maniera sequenziale;
- il metodo parallelCountNGrams() che calcola l'istogramma degli n-grams in maniera parallela.

```cpp
class NGramsCounter{
public:

    explicit NGramsCounter(int nGramLength);
    virtual ~NGramsCounter();

    void parallelCountNGrams(const std::vector<std::string>& words, int num_threads);

    void countNGrams(const std::vector<std::string>& words);

    const std::unordered_map<std::string, int> &getMap() const;

private:
    int NGramLength;
    std::unordered_map<std::string, int> map;



    std::vector<std::string> extractNGramsFromWord(const std::string& word) const;
};
```

Figure 1: Struttura classe nGramCounter

2.1 Versione sequenziale

```
void NGramsCounter::countNGrams(const std::vector<std::string>& words) {
    map.clear();
    for(const auto & word : const string & : words){
        if(word.size() >= NGramLength) {
            vector<string> ngrams = extractNGramsFromWord(word);
            for (auto & ngram : string & : ngrams) {
                if (!map[ngram]) {
                    map[ngram] = 1;
                } else {
                    map[ngram]++;
                }
            }
        }
    }
}
```

Figure 2: Versione sequenziale

Nel caso sequenziale il metodo riceve in ingresso un vettore contenente tutti i vocaboli presenti nel testo, itera su questi ultimi e per ciascuno, utilizzando il metodo extractNGramsFromWord(), determina gli n-gram presenti in ogni parola ed aggiorna i relativi contatori.

2.2 Versione Parallela

Anche nel caso parallelo il metodo riceve in ingresso un vettore contenente tutti i vocaboli presenti nel testo, tuttavia in questo caso la computazione è eseguita in una sezione parallela definita attraverso la direttiva #pragma omp parallel. Da questo momento si generano un numero di thread pari a quelli indicati nella direttiva che eseguiranno le operazioni in maniera parallela sfruttando i diversi core della CPU.
Per ciascun thread viene quindi generata una mappa locale privata che sarà utilizzata per l'esecuzione dell'accumulazione parziale dell'istogramma.
L'iterazione sull'intero dataset di parole infatti in questo caso è eseguita mediante un ciclo for parallelo che smista le diverse iterazioni tra i thread.
Infine ciascun thread dovrà accumulare il proprio risultato parziale nella mappa globale e ciò viene eseguito in una sezione critica del codice al fine di evitare eventuali scritture concorrenti con conseguente corruzione della memoria.

```
void NGramsCounter::parallelCountNGrams(const vector<std::string> &words, int num_threads) {
    int size = static_cast<int>(words.size());
    map.clear();

#pragma omp parallel num_threads(num_threads)  default(none) shared(size, words)
    {
        unordered_map<string, int> threadMap;

#pragma omp for nowait
        for (int i=0; i < size; i++) {
            if (words[i].size() >= NGramLength) {
                vector<string> ngrams = extractNGramsFromWord( word: words[i]);
                for (auto &ngram : string & : ngrams) {
                    if (!threadMap[ngram]) {
                        threadMap[ngram] = 1;
                    } else {
                        threadMap[ngram]++;
                    }
                }
            }
        }

#pragma omp critical
        for (auto [ngram : const string , count : int ]: threadMap) {
            map[ngram] += count;

        }

    }

}
```

Figure 3: Versione parallela

3 Analisi dei risultati

Al fine di valutare il beneficio offerto dalla versione parallela in termini di speed-up sono stati eseguiti diversi esperimenti. Lo speed-up è stato calcolato in termini di wall-clock time ovvero il tempo di esecuzione della funzione del calcolo dell'istogramma.

3.1 Valutazione del numero di thread

Il primo esperimento condotto è volto all'individuazione del numero di thread ottimale nel caso dell'esecuzione parallela per la macchina su cui è stato eseguito il programma (si tratta di una CPU Intel® Core™ i7-9750H a 6 core). I test sono stati condotti utilizzando come testo di riferimento David Copperfield di Charles Dickens replicato 50 volte per un totale di 17.892.500 parole. È stato calcolato il tempo di esecuzione variando il numero di thread da 1 a 24. I risultati sono mostrati nel grafico in figura 4.

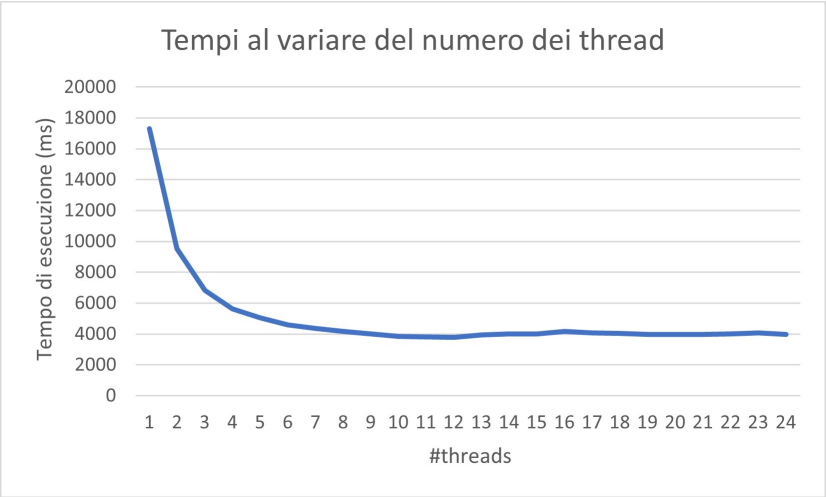

Figure 4: tempi al variare del numero di threads

Come era lecito attendersi dato che la CPU su cui sono stati eseguiti i test ha 6 core si osserva che i tempi di esecuzione scendono rapidamente (andamento esponenziale) al crescere del numero di thread fin al valore 12 oltre il quale i tempi rimangono pressoché costanti. Questo si verifica in quanto si è raggiunto il limite massimo di parallelizzazione per la macchina.

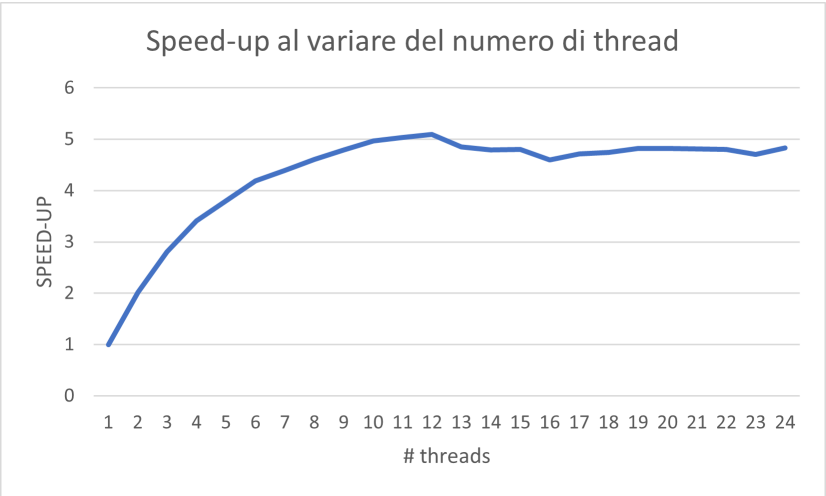

Figure 5: Speed-up del programma sequenziale rispetto al parallelo al variare del numero di threads. Si osserva il massimo speed-up che vale circa 5 per un numero di threads pari a 12.

3.2 Confronto performance algoritmi sequenziale e parallelo

In questo esperimento si confrontano le due versioni del programma valutandone lo speed-up. Sono stati eseguiti 20 esperimenti aumentando ad ogni iterazione il numero di repliche del libro in input di un numero pari a 5 e misurando i tempi di esecuzione dei due algoritmi. I risultati sono mostrati in figura 6.

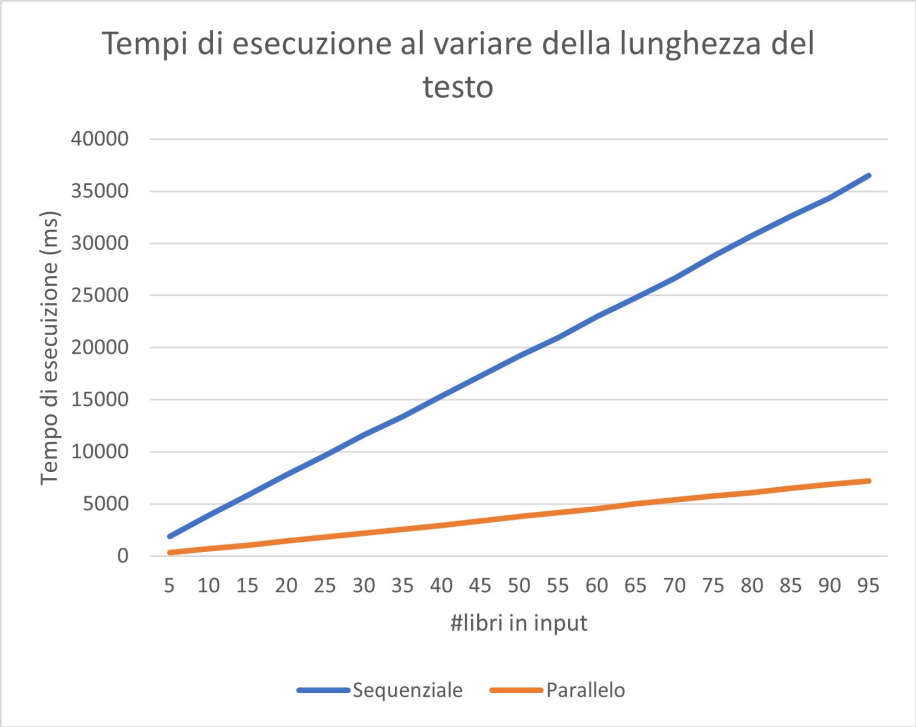

Figure 6: Tempi di esecuzione al variare del numero di libri in input

Come era lecito attendersi il programma parallelo risulta molto più rapido con uno speed-up pressoché costante ed intorno a 5.2.

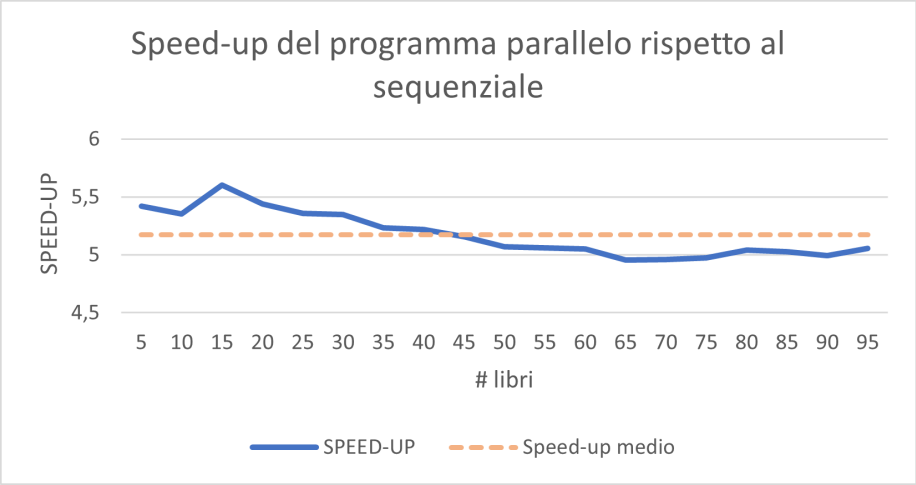

Figure 7: Speed-up del programma parallelo rispetto al sequenziale