

Conteggio di N-grams in Python

Alessandro Marinai, Eleonora Ristori

Aprile 2023

1 Introduzione

In questa relazione viene presentato l'elaborato in cui vengono implementate in Python due versioni di una funzione che ha come scopo quello di contare gli n-grams di un testo dato in input. Le versioni sono una sequenziale e una parallela che utilizza la libreria Joblib. Verrà mostrato come variano le performance di queste due funzioni al variare dei parametri di input.

2 Versione sequenziale

Le due versioni della funzione appena descritta sono state implementate come metodi della classe NGramCounter che ha come attributi un intero che rappresenta la lunghezza degli n-grams che si vuole contare ed una mappa in cui si accumula il conteggio totale delle occorrenze degli n-grams.

```
class NGramCounter:
    def __init__(self, length):
        self.nGramLength = length
        self.map = {}
```

La versione sequenziale è quella implementata nel metodo countNGrams():

```

def countNGrams(self, words):
    self.map = {}
    for word in words:
        if len(word) > self.nGramLength:
            ngrams = self.extractNGramsFromWord(word)
            for ngram in ngrams:
                if ngram in self.map:
                    self.map[ngram] = self.map[ngram] + 1
                else:
                    self.map[ngram] = 1

```

Il metodo riceve come parametro una lista contenente tutte le parole del testo di cui si vuole contare gli n-grams. Questa lista viene scorsa completamente e, ad ogni passo dell'iterazione, in primo luogo vengono estratti dalla parola tutti gli n-grams che contiene attraverso il metodo `extractNGramsFromWord()`.

```

def extractNGramsFromWord(self, word):
    ngrams = []
    for i in range(0, len(word) - self.nGramLength + 1):
        ngrams.append(word[i:self.nGramLength + i])
    return ngrams

```

Successivamente, per ogni n-gram estratto dalla parola viene incrementato il valore del corrispondente contatore nella mappa e, in caso l'n-gram non fosse ancora stato aggiunto alla mappa, viene inizializzato il relativo valore a 1.

3 Versione parallela

Questa versione suddivide la lista delle parole in slices di lunghezza `window_size` in modo da distribuire il problema di contare gli n-grams in sottoproblemi più semplici eseguibili dai processi in maniera parallela con il metodo `threadCountNgrams()`.

```

def threadCountNgrams(self, words):
    map = {}
    for word in words:
        if len(word) > self.nGramLength:
            ngrams = self.extractNgramsFromWord(word)
            for ngram in ngrams:
                if ngram in map:
                    map[ngram] = map[ngram] + 1
                else:
                    map[ngram] = 1
    return map

```

Questo metodo ha lo scopo di computare il numero di n-gram della slice che gli è passata in input, generando una mappa specifica per ogni processo in cui si accumulano i risultati parziali.

Con la chiamata alla funzione Parallel viene eseguito threadCountNgrams() in modo parallelo e le mappe in cui sono raccolti i risultati parziali vengono inserite in result. Successivamente vengono scorsi i risultati dei singoli job in maniera sequenziale per evitare di introdurre l'overhead dovuto alla presenza di una sezione critica. Ecco il metodo descritto:

```

def parallelCountNgrams(self, words, n_jobs, window_size):
    slices = [slice(start, start + window_size)
               for start in range(0, len(words) - window_size + 1, window_size)]
    slices.append(slice(len(words) - len(words) % window_size, len(words)))
    result = Parallel(n_jobs=n_jobs)(delayed(self.threadCountNgrams)(words[sl]) for sl in slices)
    self.map = {}
    for i in range(len(result)):
        for ngram in result[i]:
            if ngram in self.map:
                self.map[ngram] = self.map[ngram] + result[i][ngram]
            else:
                self.map[ngram] = result[i][ngram]

```

4 Confronto della performance tra le due versioni

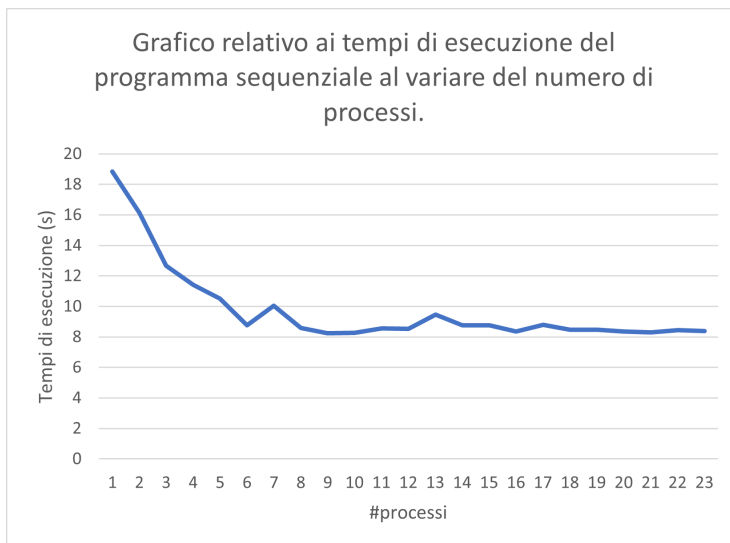
4.1 Valutazione delle performance al variare della dimensione della finestra e del numero di processi

Prima di confrontare i due metodi, è stato testato il funzionamento della versione parallela dato che questa dipende da alcuni iperparametri da cui scaturiscono performance diverse, ovvero il numero dei thread e la `window_size`.

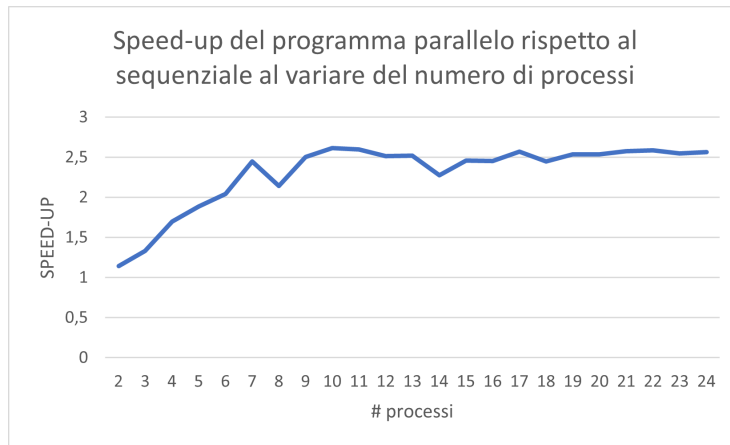
Per stabilire il valore ottimale degli iperparametri sul calcolatore utilizzato, il quale ha una CPU Intel® Core™ i7-9750H a 6 core, sono stati condotti diversi esperimenti utilizzando come input il libro David Copperfield di Charles Dickens replicato 50 volte per un totale di 17.892.500 parole.

4.1.1 Valutazione del numero ottimale di jobs

In questo primo esperimento, è stato variato il parametro `n_jobs` da 2 a 24 e sono stati memorizzati i tempi di esecuzione. Dal grafico riportato si osserva la variazione delle performance al crescere del numero dei processi, determinando quindi che il numero ottimale è proprio il numero di core del calcolatore ovvero 6, dato che una volta arrivati a 6 processi non si nota un miglioramento netto del tempo di esecuzione.

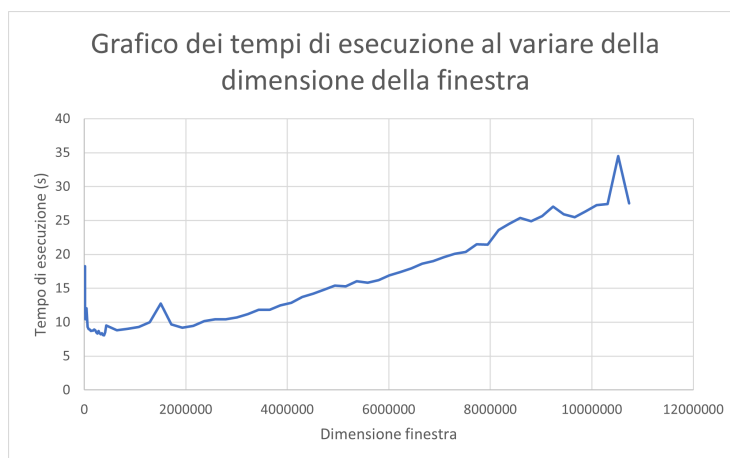


Questo secondo grafico riporta lo speed-up rispetto al variare del numero di thread. Dopo la crescita iniziale, lo speed-up si assesta attorno al valore 2.5.

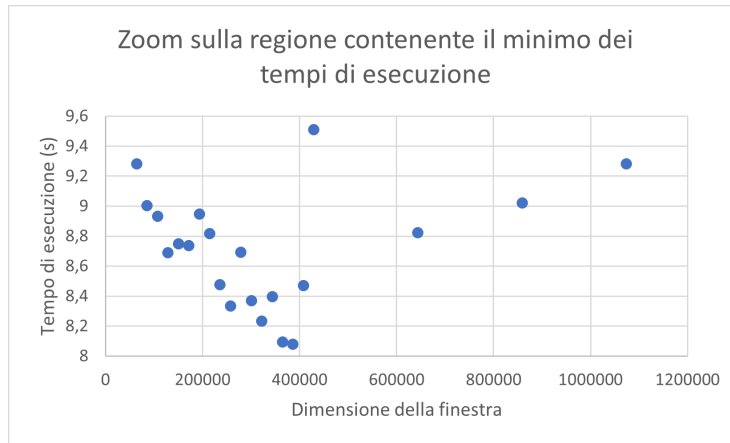


4.1.2 Valutazione della dimensione ottimale della finestra

Successivamente è stato ricercato il valore ottimale per la dimensione della finestra. Anche in questo caso sono stati memorizzati i tempi di esecuzione di diverse run successive con `window_size` variabile. Il grafico risultante da questo esperimento mostra un andamento che vede il minimo per una dimensione della finestra abbastanza piccola. Si osserva infatti un trend di crescita al crescere della dimensione fino ad arrivare al valore massimo scegliendo la dimensione della finestra massima ovvero pari alla lunghezza totale della lista di vocaboli. In quest'ultimo caso infatti il programma si comporta in maniera sequenziale. Nel grafico si osserva però anche un aumento dei tempi di esecuzione quando la dimensione della finestra risulta troppo piccola. Ciò è dovuto al fatto che in questi casi di maggior parallelismo gli overhead introdotti sono tali da provocare un peggioramento delle performance.

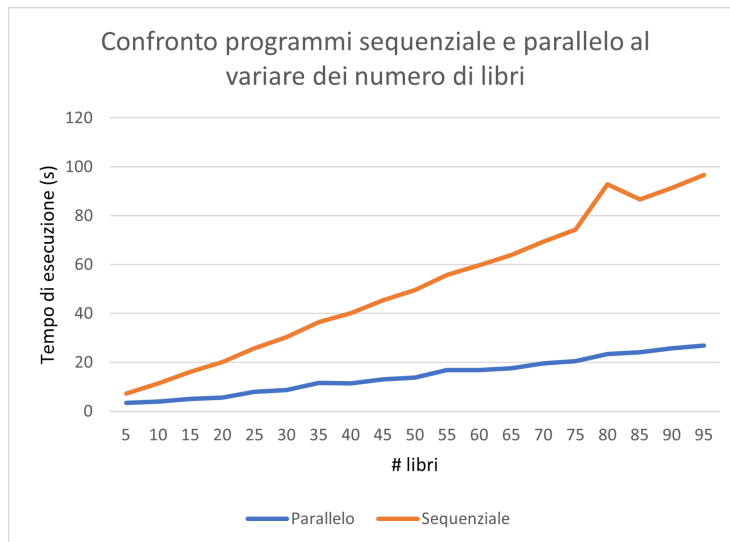


Si riporta inoltre uno zoom in corrispondenza del minimo del grafico. Da qui è possibile quindi determinare una stima della dimensione ottimale della finestra pari a circa 40000 parole.



4.2 Confronto tra la versione sequenziale e parallela

Una volta determinati gli iperparametri ottimali per l'esecuzione della versione parallela, sono stati memorizzati i tempi di entrambe le versioni al variare della dimensione dell'input, cioè in particolare al variare del numero di volte per cui viene replicato il libro David Copperfield. I risultati sono riportati nel grafico:



Nella figura sottostante si riportano i risultati relativi allo speed-up della

versione parallela rispetto a quella sequenziale che si assesta intorno al valore di 2.4 al variare della dimensione dell'input.

