
Università degli Studi di Padova

Facoltà di Ingegneria


*Corso di Laurea Magistrale in Ingegneria Informatica*

Relazione di CALCOLO PARALLELO

# Algoritmo Knuth-Morris-Pratt


*Autori*

Raffaele Di Nardo Di Maio 1204879

Cristina Fabris 1205722



Anno Accademico 2018-2019

# Indice

# 1

# Descrizione del Problema

L'algoritmo analizzato si occupa di effettuare un pattern matching. Il problema può essere formalizzato nel seguente modo: definito un insieme $\Sigma$ di simboli, detto *alfabeto*, si vuole comprendere se una sequenza di elementi dell'alfabeto

$$\{x_0\, x_1 \ldots x_{n-1} \;\mid\; x_i \in \Sigma \;\; 0 \leq i \leq n-1\}$$

detta *pattern*, sia contenuta o meno all'interno di una sequenza di dimensioni maggiori.
Nel nostro caso l'algoritmo utilizza in input una stringa alfanumerica e un testo in cui effettuare la ricerca e restituisce l'indice del carattere iniziale della prima occorrenza del pattern alfanumerico.
Nel corso degli anni sono stati studiati ed implementati diversi algoritmi che si occupano di risolvere tale problema poiché quest'ultimo si presenta in campi di ricerca di vario genere. Per citare solo un esempio, nella bioinformatica l'algoritmo viene utilizzato nella ricerca di una determinata sequenza di basi all'interno del DNA.
L'algoritmo sequenziale brute force per la ricerca di stringhe scandisce il testo, utilizzando numerose interazioni e ripetendo confronti già effettuati. Ad ogni iterazione vengono confrontati in maniera sequenziale gli elementi del testo e quelli della stringa per vedere se è presente un'occorrenza. Al primo confronto che restituisce esito negativo si passa all'iterazione successiva in cui i confronti vengono ripetuti dall'inizio del pattern.
Quest'algoritmo è poco efficiente, in quanto non tiene conto, in nessun modo, delle informazioni acquisite nelle iterazioni precedenti.
Un algoritmo più complesso ed efficiente è, invece, quello di Knuth Morris Pratt (KMP) (Sezione 2.1). Questo è stato sviluppato da Knuth e Pratt e indipendentemente da Morris nel 1975. In questo caso, l'algoritmo ha prestazioni migliori in quanto riduce il numero di confronti necessari alla ricerca. Si fa uso di una tabella che viene calcolata preliminarmente ed in essa vengono salvate informazioni riguardanti la posizione, all'interno del pattern, da cui ricominciare il confronto.
In questo modo, non tutti gli elementi del pattern precedentemente trovati nel testo devono essere riesaminati. Sfruttando tutte le informazioni acquisite precedentemente, si riesce a diminuire il numero di confronti necessari nell'approccio classico.

L'algoritmo sviluppato all'interno della trattazione è una particolare implementazione parallela dell'algoritmo KMP che permette di lavorare in contemporanea su più sezioni del testo iniziale (Sezione 2.2).
Il protocollo di progammazione multiprocessore che è stato utilizzato da questa variante dell'algoritmo di Knuth, Morris e Pratt, è basato sullo scambio di messaggi e viene chiamato Message Passing Interface (MPI).
In seguito verranno analizzate le prestazioni e le misurazioni temporali effettuate al variare del numero di processori.

# 2

# Descrizione dell'algoritmo

Il procedimento di pattern-matching sequenziale naive non sfrutta la possibile presenza di una serie di confronti con esito positivo (match), verificatasi prima di un esito negativo (mismatch) [1].
Una conseguenza di tale approccio è la dipendenza del tempo di computazione sia dalla grandezza del testo($n$) che da quella del pattern alfanumerico ($m$), rendendo nel caso peggiore le prestazioni quadratiche rispetto ad $n$. Il numero di confronti effettuati, infatti, corrisponde a $\Theta((n-m+1)m)$.
L'algoritmo utilizzato divide il testo in sezioni e ne assegna una ad ogni processore disponibile. Ognuno di questi, in parallelo rispetto agli altri, esegue una ricerca del pattern voluto con l'algoritmo KMP e restituisce l'indice della prima occorrenza trovata. Nel caso in cui il pattern sia presente all'interno del testo, verrà restituito l'indice minore tra quelli individuati dai processi; altrimenti ne verrà notificata l'assenza all'utente.

## 2.1 Algoritmo Knuth Morris Pratt

L'algoritmo KMP esegue pattern-matching in tempo lineare rispetto alla sola dimensione del testo. Sfrutta i risultati ottenuti in precedenza per evitare di dover effettuare nuovamente confronti su caratteri già analizzati. Per rispettare queste prestazioni viene eseguita inizialmente una fase di pre-processamento del pattern, utilizzata per evitare di dover compiere maggiori confronti come nel procedimento naive[1].
Per comprendere in che modo venga effettuata questa fase, è conveniente introdurre i concetti di prefisso e suffisso di una stringa. Con il termine *prefisso* di lunghezza $k$ di una stringa si intendono i primi $k$ caratteri che la compongono, mentre con *suffisso* di lunghezza $k$ gli ultimi $k$. Per ogni prefisso del pattern, viene ricercato il suo prefisso di lunghezza maggiore, che ne costituisca anche un suffisso. Il pre-processamento restituisce un vettore di numeri interi della stessa lunghezza del pattern, chiamato vettore di fallimento o fail vector. Questo contiene, in ogni cella di indice $j$, la lunghezza del prefisso ricercato, come specificato in precedenza, nella sottostringa del pattern (che si estende dal primo carattere al $j$-esimo) (Figura 2.1). In questo modo, accedendo ai valori salvati nel vettore, nel caso in cui si verifichi un mismatch, si potrà sapere quali

[1] esito positivo = corrispondenza di un carattere nel testo e un carattere nel pattern

caratteri confrontare successivamente all'interno del testo e del pattern. Una

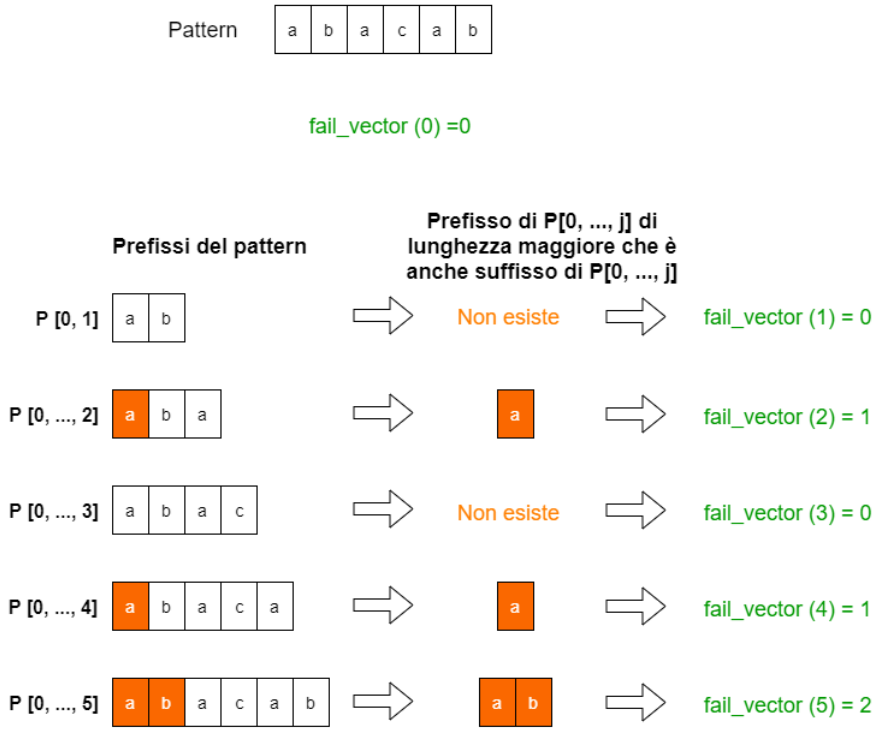


Figura 2.1: Esempio di computazione del fail vector per il pattern "abacab".

volta terminata la computazione del vettore, l'algoritmo inizia a confrontare gli elementi del testo con quelli che formano il pattern cercato. Nel caso in cui si verifichi un mismatch tra l'elemento $i$-esimo del testo e il $j$-esimo del pattern, il successivo confronto avviene tra l'elemento $t$-esimo del testo e l'elemento $k$-esimo del pattern. Gli indici $t$ e $k$ vengono calcolati con la seguente regola:

$$\begin{cases} k = fail\_vector[j-1] & t = i & se\ j \neq 0 \\ k = 0 & t = i+1 & se\ j = 0 \end{cases}$$

Una volta che i confronti hanno esito positivo per ogni carattere del pattern viene restituito l'indice iniziale dell'occorrenza del pattern nel testo.

## 2.2 Parallelizzazione

Sfruttando più processori è possibile effettuare la ricerca del pattern in più porzioni del testo contemporaneamente. In fase di esecuzione, il programma riceve in input da linea di comando il nome del file contenente il testo e il pattern da cercare.

Per prima cosa viene selezionato un processore che funga da "master" e si occupi di elaborare tutte le informazioni necessarie per l'esecuzione dell'algoritmo KMP. Nel nostro caso questo processore "privilegiato" è quello di rank 0 e i suoi compiti principali sono:

1. **Pre-processamento del pattern**
   Calcola il fail vector applicando al pattern di partenza la funzione di fallimento

2. **Elaborazione del testo** Legge in input il nome del file in cui cercare il pattern e si occupa di aprire lo stream verso di esso. Legge il contenuto del testo ed in seguito chiude lo stream verso di esso. Elabora le stringhe necessarie agli altri processori per applicare il KMP attraverso due fasi (Figura 2.2):

   - **Ciclo 1**
     Una volta salvato l'intero testo all'interno di una stringa, rimuove $m-1$ elementi da essa ( $m$ = lunghezza del pattern) e la divide in un numero di sezioni pari al numero di processi disponibili.
     Se la stringa risultante dalla rimozione degli ultimi caratteri non dovesse essere un multiplo del numero di processori utilizzati, viene svolta una ripartizione intera della stringa e gli elementi non inclusi vengono inviati al processore di $rank = numero_processori - 1$. La cosa più importante è che le stringhe debbano essere inviate rispettando l'ordine crescente dei rank e associandole in questo modo ai vari processori.
   - **Ciclo 2**
     Nel caso in cui siano presenti occorrenze del pattern a cavallo tra due sezioni del testo, calcolate nel **Ciclo 1**, non verranno rilevate. Per risolvere tale problema, il processore 0, elabora anche le sottostringhe necessarie a ciascun processore per verificare la presenza di possibili occorrenze in quelle zone.
     Le sequenze elaborate in questa fase sono costituite dagli ultimi $m-1$ caratteri di una sezione del **Ciclo 1** e i successivi $m-1$ caratteri. Da questa elaborazione del testo, si può evincere la nostra scelta di escludere alcuni caratteri nel primo ciclo.

3. **Distribuzione dei dati** Distribuisce a ogni processore le sottostringhe elaborate nel **Ciclo 1** e nel **Ciclo 2** ed il fail vector calcolato nella prima fase. Inoltre invia ad ogni processore il valore intero necessario per stabilire quale sia l'indice di un'eventuale occorrenza nelle stringhe del Ciclo 2, all'interno del testo originario.

4. **Gestione dell'output** Verifica quale sia l'indice minore tra quelli restituiti da ogni processore e mostra sullo standard output. Nel caso in cui non sia stato trovato il pattern, notifica ciò all'utente.

Tutti i processi, compreso il "master", in seguito alla distribuzione dei dati da parte del processore di rank 0, applicano l'algoritmo KMP sulle stringhe del Ciclo 1. Nel caso in cui non trovassero un'occorrenza, eseguono la stessa ricerca sulla stringhe del Ciclo 2. I risultati vengono poi inviati verso il "master" che effettuerà la gestione dell'output.

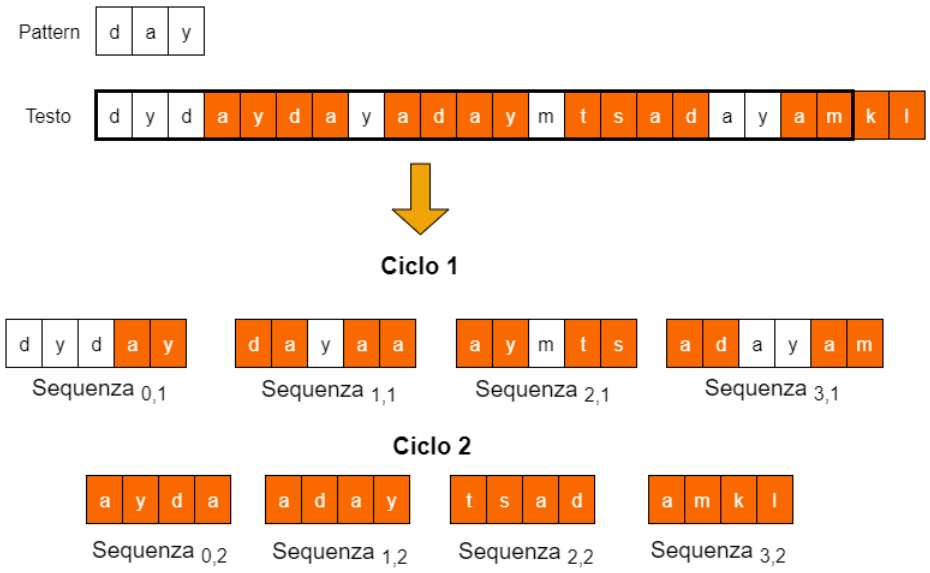


Figura 2.2: Esempio di computazione delle sequenze dei Cicli 1 e 2, utilizzando 4 processori.

# 3

# Analisi della complessità computazionale

# 4

# Analisi sperimentale

## 4.1 Tempi totali d'esecuzione e speed up

## 4.2 Tempi d'esecuzione al variare della taglia dell'input

## 4.3 Impatto delle comunicazioni

## 4.4 Test con parametri di input asimmetrici

# 5

# Conclusioni

# Bibliografia

[1] https://www.studocu.com/it/document/universita-degli-studi-di-napoli-parthenope/programmazione-ii-e-laboratorio-di-programmazione-ii-cfu-9/appunti/16-algoritmo-kmp-per-lo-string-matching/1519440/view