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. In forma generale questo problema può essere formalizzato nel seguente modo: definito un insieme Σ di elementi, detto *alfabeto*, trovare un'occorrenza di una sequenza di elementi

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

detta *stringa*, all'interno di un più ampio testo formato sempre da elementi appartenenti all'alfabeto.
Nel nostro caso l'algoritmo riceve in input la stringa e un testo e restituisce come output l'indice corrispondente alla posizione del primo carattere della prima occorrenza in cui è possibile trovare, all'interno del testo, la stringa data.
Negli anni sono stati studiati ed implementati diversi algoritmi che si occupano di questa ricerca poiché si tratta di un problema che trova applicazioni negli ambiti più vari. Per citare solo un esempio, nella bioinformatica è applicato nella ricerca di una determinata sequenza di basi all'interno del DNA.
L'algoritmo brute force per la ricerca di stringhe scandisce ripetutamente il testo, ogni volta iniziando dall'elemento successivo a quello utilizzato all'inizio dell'iterazione precedente. Ad ognuna di queste confronta gli elementi del testo e quelli della stringa per vedere se è presente un'occorrenza. Al primo confronto che restituisce esito negativo passa all'iterazione successiva.
Quest'algoritmo è poco efficiente, in quanto non tiene in nessun modo conto delle informazioni acquisite dai confronti nelle iterazioni precedenti a quella attualmente in esecuzione.
Un algoritmo più complesso ed efficiente è, invece, quello di Knuth Morris Pratt (KMP). Questo è stato sviluppato da Knuth e Pratt e indipendentemente da Morris nel 1975. La maggior efficienza si verifica grazie al fatto che cerca di diminuire il numero di confronti necessari alla ricerca. Infatti, sfrutta una tabella che viene computata all'inizio del procedimento, nella quale sono salvate le posizioni successive a quella corrente, in cui è possibile trovare un'occorrenza della sequenza cercata.
In questo modo gli elementi del testo precedentemente verificati non vengono ricontrollati, sfruttando così tutte le informazioni acquisite precedentemente e diminuendo di molto il numero di confronti rispetto all'algoritmo brute force.
L'algoritmo che andremo ad analizzare è una particolare implementazione dell'algoritmo KMP che sfrutta la ricerca in parallelo su più sezioni del testo iniziale.

Infatti, divide quest'ultimo in più parti (pari al numero di processi che si vogliono attuare in parallelo) e all'interno di ogni set di elementi dell'alfabeto che si vengono a creare cerca la stringa richiesta. Inoltre, per la ricerca di occorrenze a cavallo tra più set, applica una particolare accortezza, sfruttando anche in questo caso il lavoro di più processi in parallelo.

2

Descrizione dell'algoritmo

L'algortimo utilizzato divide il testo in sezione e ne assegna una ad ogni processore disponibile. Questi eseguono parallelamente una ricerca del pattern voluto con l'algoritmo KMP e restituiscono l'indice della prima occorrenza trovata. Infine, uno dei processi verifica qual è l'indice complessivo minore e lo restituisce all'utente.

2.1 Algoritmo Knuth Morris Pratt

L'algoritmo KMP esegue pattern-matching in tempo lineare nella dimensione del testo in cui cercare il pattern. Infatti, sfrutta i risultati ottenuti precedentemente per evitare di dover rifare confronti su elementi già analizzati. Per fare questo esegue una fase di preprocessamento del pattern prima della ricerca vera e propria. Questa viene sfruttata nel caso in cui si verifichi un confronto con esito negativo (mismatch) dopo una serie di successi (match).
In pratica, considerando una sottostringa del pattern, lo si analizza alla ricerca del maggior prefisso che costituisce anche suffisso della stessa. Con il termine *prefisso* si intendono i primi n caratteri che compongono una stringa, mentre con *suffisso* gli ultimi n. Tale funzione restituisce un vettore della stessa lunghezza del pattern, contenete, per ogni indice j, la lunghezza del maggior prefisso che rispetti la proprietà descritta, relativamente alla sottostringa che va dall'inizio del pattern al suo j-esimo elemento. In questo modo, nel caso in cui si verifichi un mismatch tra testo e pattern, si potrà sapere di quando traslare quest'ultimo rispetto al testo accedendo ai valori salvati nel vettore.

(INSERIRE IMMAGINE CON ESEMPIO DI COMPUTAZIONE DI FAIL)

Una volta terminata la computazione del vettore, l'algoritmo passa ad eseguire il pattern matching. Confronta gli elementi del testo con quelli che formano il pattern cercato e, nel caso in cui si verifichi un mismatch tra l'elemento i del testo e il j-esimo del pattern, riprende la ricerca confrontando l'elemento t-esimo del testo con l'elemento k-esimo del pattern. Gli indici t e k vengono calcolati con la seguente regola: (MODIFICARE IN UNA FORMA PIÙ CARINA)
- se j!=0: k = valore contenuto nella posizione j-1 del vettore precalcolato e t =i

- se j=0: k=0 e t = i+1
Una volta che i confronti raggiungono con successo la fine del pattern viene restituito l'indice iniziale dell'occorenza appena trovata.

(INSERIRE RIFERIMENTO ALL'INDIRIZZO WEB
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)

2.2 Parallelizzazione

Sfruttando più processori è possibile effettuare la ricerca del pattern in più parti del testo contemporaneamente.
Per prima cosa viene scelto un processore che si occupi di computare tutte le informazioni necessarie per l'esecuzione dell'algoritmo KMP. Nel nostro caso questo processore è lo zeresimo (ESISTE COME PAROLA?).
Questo acquisisce il file in cui effettuare la ricerca ed esegue il preprocessamento del pattern (SI CAPISCE COSA INTENDO? FAIL). Poi divide il file in un numero di sezioni pari al numero di processi disponibili, con l'accortezza di non considerare gli ultimi *m-1* elementi (con m = lunghezza del pattern). Nel caso in cui il numero di elementi rimanenti non sia divisibile per il numero di processi, aggiunge quelli restanti all'ultima sezione. Una volta effettuati questi procedimenti, comunica a tutti i processi la loro sezione di testo che da analizzare e il vettore di indici precomputato.
In questo modo però, nel caso in cui siano presenti occorrenze del pattern a cavallo tra due sezioni del testo, non vengono trovate. Per evitare ciò, ogni processo effettua una seconda ricerca, sempre attraverso l'utilizzo dell'algortimo KMP, su una nuova sezione del testo inviatagli dal processo 0. Questa è formata dagli ultimi *m-1* elementi di una sezione e dai primi *m-1* di quella successiva. L'ultimo processo, invece, analizza gli ultimi *m-1* elementi dell'ultima sezione e gli *m-1* che erano stati precedentemente esclusi dalla suddivisione del testo. In questo modo tutti i processi si ritrovano a dover lavorare anche durante il secondo ciclo, cosa che invece non si sarebbe verificata se il file fosse stato interamente diviso all'inizio.
Una volta che tutti i processi hanno terminato la ricerca, il processo 0 verifica qual è l'indice minore tra quelli restituiti. Questo è l'indice corrispondente della prima occorenza del pattern all'interno del testo.

3

Analisi della complessità computazionale

4

Analisi sperimentale

5

Conclusioni

Bibliografia