Alessandro Bitocchi
Matricola: **1837174**
Sapienza Università di Roma

Relazione

SCELTE PROGETTUALI

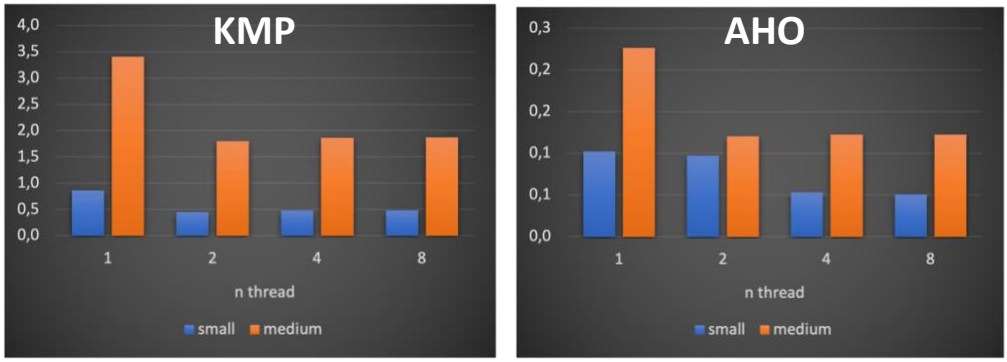
Per ciascun algoritmo proposto ho sviluppato la versione seriale, nel caso dell'algoritmo Aho-Corasick ho fatto riferimento al software proposto dall'utente di GitHub Morenice. Se non viene specificato al compilatore l'uso della libreria OpenMP allora viene compilata la versione seriale, il cui eseguibile si trova nella cartella bin/serial. Inizialmente avevo scritto due versioni per ciascun algoritmo (una seriale ed una parallela), ho deciso quindi di ridurre le versioni ad una ciascuno, controllando appunto l'uso della libreria. Questi file sorgente si trovano nella cartella **src.**
Per testare il software ho estratto dei pacchetti da Wireshark, ed ho creato dei file di diverse dimensioni ripetendo il campione estratto. Mentre per estrarre i pattern da ricercare ho scritto un piccolo script python che aggiunge ogni parola dal campione di pacchetti ad un insieme e successivamente modifica alcune stringhe, in modo da non permettere a tutte le parole da ricercare di fare match. Ogni volta che viene eseguito lo script python i file presenti nella cartella pattern e cambiano. Nella cartella specialCase ci sono tre file di varie dimensioni in cui le parole da ricercare non sono state cambiate, quindi tutte fanno match.
Sono state create inoltre delle dimostrazioni nella cartella **run** che testano gli algoritmi con input di varie dimensioni.
Entrambi gli algoritmi condividono una funzione comune che gli permette di creare una array di stringhe per ogni file dato in input, dunque una volta eseguita questa funzione per il file delle stringhe da ricercare e per il file dei pacchetti in cui cercare le stringhe, avrò due array di stringhe, uno contenente i pattern e l'altro contente i pacchetti. Questa funzione si trova nel file **lib/src/reader.c**.
Per parallelizzare gli algoritmi ho scelto di distribuire le stringhe da cercare ed il numero di pacchetti in cui cercarle per un dato numero di thread tramite la clausola:

```
#pragma omp parallel for num_threads(thread_count) \
reduction(+: total_match)
```

Il tempo viene cronometrato solo per la ricerca delle stringhe nel file ed è misurato in secondi.

RISULTATI OTTENUTI

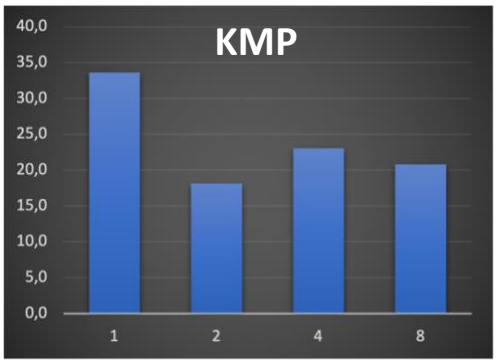
L'algoritmo Knuth Morris Pratt impiega più tempo rispetto ad Aho-Corasick.
Eseguendo entrambi gli algoritmi con input di **piccole** e **medie dimensioni** otteniamo i seguenti risultati:

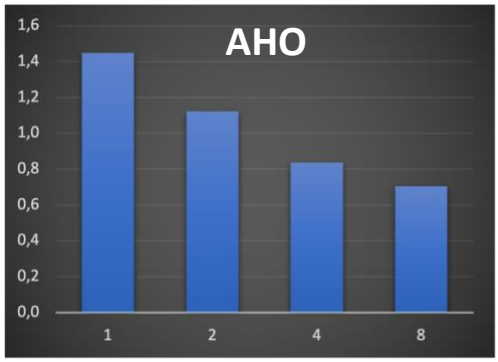


Possiamo notare che eseguendo gli algoritmi con due thread dimezziamo quasi il tempo di esecuzione, ma aumentandone il numero non riusciamo più ad ottenere un miglioramento in termini di tempo.

Alessandro Bitocchi
Matricola: **1837174**
Sapienza Università di Roma

Mentre eseguendo entrambi gli algoritmi con un input di **grandi dimensioni** otteniamo:





Notiamo che il comportamento dell'algoritmo KMP rimane inalterato, mentre AHO dimostra che aumentando il numero di thread riusciamo a migliorare il tempo di esecuzione.

Di seguito riporto le tabelle con tutti i dati raccolti:

AHO					
	input	*n thread*			
		1	*2*	*4*	*8*
	small	0,102538	0,097354	0,053333	0,050697
	medium	0,226361	0,120498	0,122231	0,122631
	large	1,449278	1,122396	0,835999	0,704457
Small	**Speedup**	1,0	1,1	1,9	2,0
	Efficiency	1,0	0,5	0,5	0,3
Medium	**Speedup**	1,0	1,9	1,9	1,8
	Efficiency	1,0	0,9	0,5	0,2
Large	**Speedup**	1,0	1,3	1,7	2,1
	Efficiency	1,0	0,6	0,4	0,3
Dati ottenuti con i pattern contenuti nella cartella samplePatterns. Valori esprimono il tempo impiegato dall'algoritmo in **secondi**					

KMP					
	input	*n thread*			
		1	*2*	*4*	*8*
	small	0,859593	0,449363	0,480194	0,477661
	medium	3,405637	1,795667	1,866720	1,871783
	large	33,640918	18,129693	23,039339	20,821137
Small	**Speedup**	1,0	1,9	1,8	1,8
	Efficiency	1,0	1,0	0,4	0,2
Medium	**Speedup**	1,0	1,9	1,8	1,8
	Efficiency	1,0	0,9	0,5	0,2
Large	**Speedup**	1,0	1,9	1,5	1,6
	Efficiency	1,0	0,9	0,4	0,2
Dati ottenuti con i pattern contenuti nella cartella samplePatterns. Valori esprimono il tempo impiegato dall'algoritmo in **secondi**					