SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica

DNA Sequence

Autori:
Simone Lidonnici - (2061343)
Marco Casu - (2041262)

19 gennaio 2025

1 Introduzione

Le parti principali di codice da parallelizzare sono la creazione della sequenza e la ricerca dei pattern. Per sequenze medio-piccole quest'ultima compone la maggior parte del tempo di compilazione, mentre aumentando la grandezza della sequenza (nell'ordine dei miliardi), la maggior parte del tempo è richiesto per generare la sequenza. Da notare che nel programma sequenziale il tempo non conta la generazione di quest'ultima mentre nei programmi paralleli (MPI, OpenMP, CUDA e MPI+OpenMP) si, cosa che causa una diminuzione dello speedup.

1.1 Test e calcolo dello speedup

Le varie versioni del programma sono state testate sul cluster eseguendo 10 test e controllando il tempo medio di essi, in particolare il test è stato eseguito con i seguenti parametri:

```
10000 0.35 0.2 0.25 0 0 0 10000 9000 9000 50 100 M 4353435
```

2 MPI

Nel programma MPI abbiamo diviso la ricerca dei pattern uniformemente tra i rank, cioè se ci sono t processi ognuno ricercherà $\frac{n}{t}$ pattern, con n numero totale di pattern (sia sample che random). Per avere poi il valore corretto dei pattern trovati e dei pattern in ogni punto della sequenza vengono eseguite delle `MPI_Reduce` su `seq_matches` e `pat_found`. Per quanto riguarda la generazione della sequenza, essa è intrinsecamente più lenta del sequenziale, anche con un solo processo MPI, per otimizzarla abbiamo tentato due opzioni. La prima è quella di far generare la sequenza solo ad un processo e poi eseguire una Broadcast, cosa che si è rivelata più veloce della versione iniziale ma comunque più lento del sequenziale. La seconda opzione, presente nel file finale, è stata quella di dividere la sequenza tra i vari rank, nello stesso modo in cui vengono divisi i pattern e poi eseguire una `MPI_Allreduce` per far avere a tutti i rank la sequenza completa. Data la natura puramente sequenziale delle funzioni rng, ogni rank esegue un `rng_skip` per poter iniziare a generare i suoi numeri random da un punto avanzato della sequenza.
Di seguito si può vedere il grafico dei tempi in relazione al numero di processi MPI, i test sono stati eseguiti con il numero massimo di processi su un nodo (32) e aumentando i nodi progressivamente: In seguito è riportato anche il grafico dello speedup: Se si vogliono consultare le informazioni in maniera più pratica, si può controllare la seguente tabella:

3 OpenMP

Nel programma OpenMP per quanto riguarda la ricerca dei pattern abbiamo creato delle variabili `pat_matches` e `seq_matches` private per ogni thread e abbiamo parallelizzato solamente il ciclo più esterno, tramite un `#pragma omp for`, questo perché i tre cicli non potevano essere collassati in uno unico data la presenza di `break` al loro interno. Dopo il ciclo for c'è una parte che viene eseguita da un thread alla volta tramite un `#pragma omp critical` che somma i valori di `pat_matches` e `seq_matches` per avere i valori finali corretti. Per quanto riguarda invece la creazione della sequenza abbiamo usato lo stesso approccio di MPI, cioè dividerla in parti uguali in base all'id del thread e far eseguire ad ogni thread `rng_skip` fino al punto dove deve iniziare a generare i propri numeri random. Per impostare il numero di thread è stato aggiunto un argomento in input, in modo da poterli impostare da linea di comando.
Di seguito si può vedere il grafico dei tempi in relazione al numero di thread, i test sono stati eseguiti aumentando il numero di thread progressivamente: In seguito è riportato anche il grafico dello speedup: Se si vogliono consultare le informazioni in maniera più pratica, si può controllare la seguente tabella: