Implementazione algoritmo SMITH-WATERMAN

Dato l'algoritmo che utilizza la CPU assegnatomi, ho realizzato un'implementazione con la GPU.

Il codice mantiene i valori trovati dall'algoritmo CPU originale, calcola e mantiene i corrispondenti valori dell'implementazione con la GPU per poi confrontarli con i valori originali.

Inoltre stampa, oltre alle tempistiche dei calcoli eseguiti dalla CPU (presenti già nell'algoritmo originale), le tempistiche del codice elaborato dalla GPU che risultano notevolmente inferiori.

1. Nel main prima del lancio del kernel

   Date le matrici *query, reference, simple_rev_cigar* e l'array *res*, ho allocato dinamicamente i corrispondenti array *Q* (di dimensione (righe di *query*)*(colonne di *query*)) , *R* (di dimensione: (righe di *reference*)*(colonne di *reference*), *RES* (di dimensione uguale alla dimensione di *res*) e *SRC* (di dimensione (righe di *simple_rev_cigar*)*(colonne di *simple_rev_cigar).*

   Prima del *get_time()* di *start_cpu*, man mano che la funzione *random* andava a riempire le matrici *query* e *reference* ne ho ricopiato i valori rispettivamente negli array *Q* e *R,* creando così due array contenenti i valori delle stringhe da confrontare che ho potuto facilmente passare per indirizzo al kernel.

   Dopo il *get_time()* di *end_cpu*, e la stampa del tempo impiegato dalla CPU (presenti nel codice dato), ho allocato con la *cudaMalloc* gli array g*query, greference, gres* e g*simple_rev_cigar* delle dimensioni rispettivamente di *query, reference, res* e *simple_rev_cigar*, e gli array gsc e *gdir* delle dimensioni di (righe di *query*)*(righe di *sc_mat*)*(colonne di *sc_mat*) e (righe di *query*)*(righe di *dir_mat*)*(colonne di *dir_mat*).
   Ho poi copiato da host a device il contenuto di Q e R in *gquery* e *greference*.

   Ho scelto di utilizzare 1000 blocchi della Grid e per ogni blocco tutti i threads disponibili ovvero 1024.
   Dopodiché ho preso il tempo con il *get_time()* della *start_gpu*, lanciato la funzione *inplinGpu* che implementa con la GPU tutto ciò che è contenuto tra *start_cpu = get_time()* e *end_cpu = get_time()* nella CPU.
   La funzione *inplinGpu* opera contemporaneamente con 1000 blocchi e per ognuno 1024 threads e passa per indirizzo gli array: *gquery, greference, gsimple_rev-cigar, gres, gsc e gdir.*
   Dopo la *get_time()* della *end_gpu*, la funzione stampa le tempistiche della GPU.
   Per poter confrontare gli arrays contenenti i risultati dei calcoli della CPU e quelli della GPU ho copiato da device a host sugli arrays precedentemente allocati *RES* e *SRC*, gli array gres e *gsimple_rev_cigar* che contengono i risultati dell'elaborazione della GPU.

2. Funzione inplinGpu

La funzione *inplinGpu* riceve, passati per indirizzo gli arrays *gquery, greference, gsimple_rev_cigar, gres* e gli arrays *gsc* e *gdir*, che all'interno della funzione prendono i nomi *sc_mat* e *dir_mat*.
Quello che ho fatto da qui in avanti è stato prendere ogni singola azione compiuta dal codice scritto per la CPU e provare a trovare dei modi perché alcune sequenze di calcoli potessero essere svolte in parallelo e indipendentemente dalle altre in modo da poterle affidare a threads diversi.

Poiché l'intero *max* della CPU è inizializzato a *ins* = -2, ho creato un array *shared* (*max*) di dimensione 1025 e ne ho parallelamente inizializzato a -2 il contenuto di ogni casella. Ho anche creato un array *shared* (*posizioni*) di dimensione analoga. Entrambi gli arrays verranno utilizzati in un secondo momento.

La CPU cicla per ogni riga delle matrici *query* e *reference* contenenti l'input randomico una serie di operazioni che porteranno a scrivere all'interno della casella n del res corrispondente all'ennesima riga delle matrici *query* e *reference*.
Ho assegnato le operazioni da svolgere su ognuna delle 1000 righe al corrispondente blocco.

Gli arrays *sc_mat* (e *dir_mat*) della GPU sono di lunghezza pari a tutte le matrici *sc_mat* (e *dir_mat*) della CPU. Ogni blocco ha accesso solo ad una sezione di caselle lungo l'array equivalenti al numero di caselle corrispondenti alla dimensione delle matrici della CPU. Come nella CPU ho inizializzato il contenuto di *sc_mat* e *dir_mat* a 0. L'ho fatto assegnando a ogni thread una uguale porzione delle caselle destinate al suo blocco, per entrambi gli arrays.
Per quanto riguarda i calcoli che portano al riempimento di *sc_mat* e *dir_mat*, ho parallelizzato in questo modo. Immaginandolo riferito ad una matrice, ogni casella di *sc_mat* e *dir_mat* determina il proprio valore in base al valore delle 3 caselle che le stanno adiacenti a sinistra, in alto e diagonalmente in alto a sinistra. Per questo ho potuto in parallelo svolgere i calcoli sulle anti diagonali notando che la somma delle coordinate per ogni anti diagonale è la stessa e aumentandola progressivamente fino ad aver calcolato i valori di tutte le celle della matrice. I calcoli eseguiti sono i medesimi che già si trovano nel codice CPU, solo rielaborati per essere eseguiti su arrays invece che su matrici. Il codice CPU per ogni casella elaborata faceva un confronto con il già esistente massimo per trovare il valore massimo tra tutti quelli della *sc_mat*. Questo non è stato possibile svolgendo i calcoli in parallelo quindi in un secondo momento ho, come per l'inizializzazione a 0 delle matrici precedenti, assegnato ad ogni thread del blocco un pezzo di *sc_mat* perché ne trovasse il massimo e lo riportasse nell'array *max* per essere poi confrontato con i massimi trovati dagli

altri threads. Contemporaneamente nell'array posizioni venivano riportate le posizioni dei *max* trovati dai thread. Per trovare il massimo tra i valori di *max* ho utilizzato un solo array. Ho infine assegnato il valore del massimo di *max* alla casella di gres la cui posizione corrispondeva all' ID del blocco.
Dopo aver sincronizzato i threads ho chiamato la funzione *backtraceGpu*.

3. Funzione backtraceGpu

Questa funzione ha richiesto una elaborazione molto più complessa rispetto alla sua corrispondente per la CPU. A *backtraceGpu* vengono passati, l'indirizzo della sezione di *gsimple_rev_cigar* su cui la funzione dovrà scrivere i risultati che poi andranno confrontati con quelli delle CPU, l'array *dir_mat*, la posizione del massimo trovato in *sc_mat* e la quantità massima di caselle dell'array *gsimple_rev_cigar* su cui la funzione potrà operare. Dato un intero *shared* partenza ho calcolato la posizione del massimo tra i *max*, passato alla funzione che lo vede con il nome *maxind*, come se la funzione vedesse solo la sezione dell'array *dir_mat* su cui può operare e non l'intero array.

Per riuscire a eseguire qualcosa in parallelo ma al contempo evitare di fare troppi calcoli inutili ho optato per la seguente soluzione: immaginando *dir_mat* come una matrice, ho operato parallelamente su una sezione di *dir_mat* immaginabile come un quadrilatero con angolo in basso a destra corrispondente alla casella partenza e di misure standard 32*32 in modo da utilizzare tutti i 1024 thread, ma con la possibilità di restringere un lato (e allungarne l'altro perché vengano utilizzati il massimo numero possibile di thread) nel caso le caselle a disposizione lungo gli assi fossero meno di 32.
I calcoli eseguiti sono, in una prima parte il calcolo della casella a cui la casella che si sta analizzando rimanda e la sua scrittura in un array *shared next* di dimensione 1024. Questo tenendo conto di alcune eccezioni: se la casella si trova sul bordo sinistro o superiore del parallelepipedo e per il suoi valori rimanda ad una casella che non è contenuta nel parallelepipedo, invece della posizione della casella a cui rimanda vengono scritti nell'array *next* i numeri simbolici 1024, 1025 o 1026 che daranno in seguito indicazioni su dove posizionare la nuova casella partenza. Un'altra eccezione è costituita dal valore 0 del contenuto della casella. In questo caso in *next* verrà segnato -1.

Dopo aver riempito l'array *next* e sincronizzato i threads inizia la fase di scrittura in *gsimple_rev_cigar*. Partendo dalla casella di partenza, seguendo le direzioni date di volta in volta dai valori dell'array *next*, ci si sposta in *next* riportando in *gsimple_rev_cigar* i valori corrispondenti alle posizioni indicate in *next*. Nel caso si trovasse in *next* il valore -1 allora la sequenza si ferma e la funzione si conclude, scrivendo uno 0 nella prima casella libera di *gsimple_rev_cigar*. Questo 0 non ha alcuna utilità nella funzione ma mi servirà nella fase di verifica.

Se invece la condizione per cui la sequenza di scrittura in *gsimple_rev_cigar* si interrompe è perché si è raggiunto il bordo del parallelepipedo allora in base al valore simbolico riportato della casella di *next*, si individua la nuova posizione partenza da cui creare un altro parallelepipedo che opererà esattamente come il precedente.
La creazione di parallelepipedi continuerà finché in *dir_mat* non si incontrerà uno 0, cosa che accadrà in condizione limite lungo il bordo sinistro e superiore della matrice immaginata a partire dall'array *dir_mat*.

Ogni blocco opererà simultaneamente per compilare *gsimple_rev_cigar* come *simple_rev_cigar* calcolando i valori della riga di *simple_rev_cigar* corrispondente al proprio IDBlocco.

4. Verifica dei risultati

Per verificare i risultati ho utilizzato il metodo più basico che conosco (anche perché non ne conosco altri e la *strcmp* non so se si adatta ai casi che devo verificare).
Per verificare la corrispondenza tra res e RES (i cui valori sono stati copiati da device a host da gres) ho verificato che facendo girare un ciclo for che si interrompesse quando superava la dimensione di res oppure quando trovava differenze tra i valori dei due array, il ciclo procede fino alla fine dell'array.
Così ho verificato la corrispondenza di res.
Per verificare la corrispondenza di *simple_rev_cigar* essendo una matrice scritta solo parzialmente, ho inserito alla fine di ogni riga scritta un valore 0 che indicasse che da lì in poi i valori non essendo inizializzati non vanno considerati.
Ho poi confrontato i valori della matrice *simple_rev_cigar* con quelli dell'array *gsimple_rev_cigar* chiedendo di segnalarmi un errore in caso di valori discordanti.
Così ho verificato la corrispondenza di simple_rev_cigar.