Smith Waterman speed up in CUDA

Lorenzo Basile

June 2023

1 Introduzione

Lo scopo del progetto è velocizzare l'algoritmo Smith Waterman su GPU, a partire da un implementazione data in C, che genera in modo randomico le sequenze di DNA da analizzare (i.e. query e reference). Il codice fornito genera N (1000) sequenze lunghe S LEN (512). Ogni sequenza della query è comparata con il reference e l'algoritmo fornisce come risultato un punteggio e il conseguente backtracking.

2 Smith Waterman

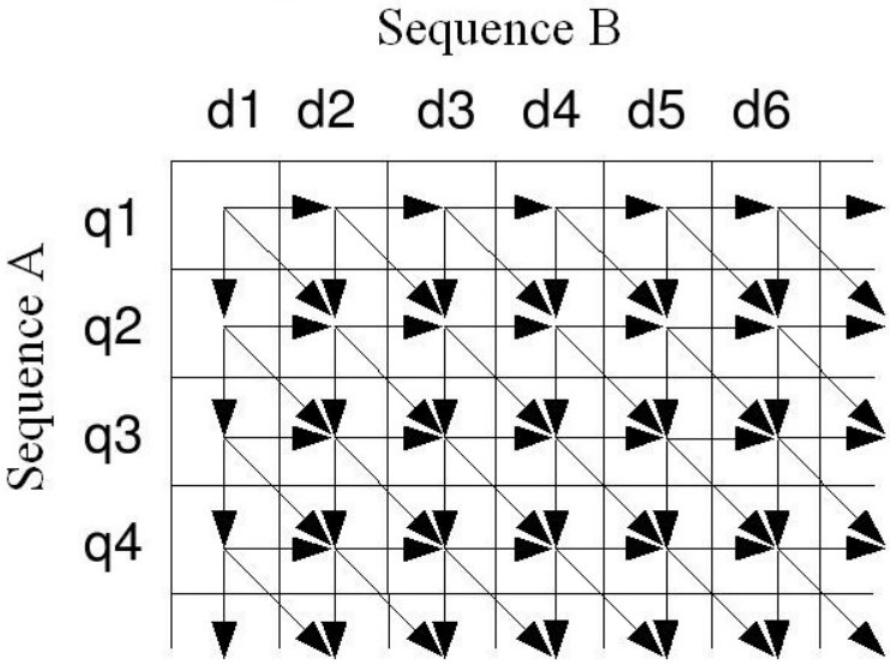
L'algoritmo Smith Waterman è un algoritmo usato nel campo della bioinformatica, con lo scopo di allineare sequenze nucleotidiche o proteiche. L'output dell'algoritmo è uno score, ovvero un punteggio che viene fornito alle due sequenze: questo indica il grado di similarità tra le due sequenze.
L'algoritmo usa una matrice che viene riempita in modo dinamico per calcolare lo score. Data la lunghezza della reference, n, e la lunghezza della query, m, la matrice ha dimensione *n+1, m+1*.
A fronte di una prima inizializzazione, dove la prima riga e la prima colonna vengono settate a zero, le seguenti celle vengono calcolate tramite la seguente formula:

$$H(i,j) = \max \begin{Bmatrix} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \text{Match/Substitution} \\ H(i-1,j) + w(a_i,-) & \text{Deletion} \\ H(i,j-1) + w(-,b_j) & \text{Insertion} \end{Bmatrix}, 1 \leq i \leq m, 1 \leq j \leq n$$

Nell'implementazione fornita il valore di match = 1, substitution = -1, deletion = insertion = -2
Nell'estensione dell'algoritmo fornita, si vuole computare anche il backtracking, ovvero in output viene fornita anche una stringa, detta *cigar*, che indica quali sono state le operazioni svolte tra le due stringe (inserzione, cancellazione, sostituzione o match).

3 Come massimizzare il parallelismo

Prima di iniziare l'implementazione in CUDA, ho analizzato le dipendenze tra i dati in modo da ottimizzare il grado di parallelismo.
Anallizzando la formula per computare la singola cella, si può facilmente vedere che ogni cella è dipendente dal valore della cella superiore, della cella alla sua sinistra e della cella in alto a sinistra, come mostrato nella foto.



Per minimizzare l'interleaving dell'hardware e massimizzare così il grado di parallelismo, si può procedere computando quindi i valori lungo le antidiagonali, eliminando di conseguenza le dipendenze tra i dati.

4 Implementazione in CUDA

Ho deciso di schedulare N blocchi e S LEN thread. In questo modo ho eseguito un multi allineamento in parallelo, ovvero ogni blocco esegue l'allineamento di una sequenza. Al termine dell'esecuzione avremo quindi allineato N sequenze in parallelo.
All'interno di ogni blocco poi, ogni thread si occupa di calcolare il punteggio della cella della matrice del punteggio.
La query e la reference vengono copiate tramite cudaMemCpy sulla global memory della GPU e viene allocato lo spazio per ospitare i risultati fornito dal codice,

ovvero un intero per ogni blocco. Viene allocato spazio anche per la cigar e per la matrice delle direzioni, che indica da dove si è ottenuto il risultato della cella.
Il codice del kernel si occupa prima di tutto di inizializzare le strutture dati.
Successivamente ogni thread computa il risultato delle celle della matrice a lui deputate: il Thread 0 computa tutti i punteggi della prima colonna, il Thread 1 si occupa della seconda colonna ... il thread S Len -esimo computa i valori dell'ultima colonna. La prima colonna e la prima riga non vengono computate perchè sono costantemente a zero.
La matrice dei risultati dovrebbe risiedere sulla shared memory, ma la sua dimensione è effettivamente troppo grande per poter essere allocata. Dato che l'algoritmo procede lungo le diagonali, e i valori necessari si trovano solamente sulle ultime due diagonali, ho quindi ridotto la matrice memorizzando solamente le ultime due antidiagonali, iterativamente.

	0	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	T0	T1	T2	T3
2	0	T0	T1	T2	T3
3	0	T0	T1	T2	T3

Nella figura di esempio, alla prima iterazione solo il thread 0 è in grado di computare il risultato, alla second iterazione possono lavorare sia T0 che T1 etc...
Il massimo grado di parallelismo è ottenibile solo sulla antidiagonale maggiore, in cui tutti i treadh schedulati possono lavorare in contemporanea.
Per evitare problemi di sincronismo, ogni thread salva su un array allocato in shared memory il proprio risultato migliore. Alla fine della computazione della matrice, il Thread 0 scorre l'array con i risultati migliori per ottenere il miglior risultato globale del blocco, e lo scive nell'array dei risultati in global memory.
Solamente il thread 0 poi esegue il backtracking in cui, a partire dalla cella con il risultato migliore e la matrice delle direzioni, ricostruisce la cigar.

5 Conclusioni e possibili miglioramenti

Il collo di bottiglia dell'algoritmo originale è la computazione della matrice, che viene parallelizzata con l'implementazione svillupata.
Tuttavia il codice presenta ancora dei possibili punti di miglioramento.L'ultima

parte dell'algoritmo è eseguita solamente da un thread, mentre idealmente sarebbe possibile rendere anche quella in esecuzione parallela, infatti la ricerca del massimo all'interno di un array è facilmente implementabile in CUDA. Anche la parte del backtracking potrebbe essere svolta su più thread.
In linea generale questi sono i risultati medi ottenuti durante la fase di testing.