Fattorizzazione QR: algoritmo di Gram-Schmidt modificato

Sia $A \in \mathbb{R}^{m \times n}$, con $m \geq n$, allora esistono una matrice ortogonale $Q \in \mathbb{R}^{m \times n}$ e una matrice triangolare superiore $R \in \mathbb{R}^{n \times n}$, tali che:

$$A = QR$$

L'algoritmo di Gram-Schmidt prevede di considerare il set di vettori composto dalle colonne di A e di fornirne una sua ortonormalizzazione. I vettori così calcolati andranno a comporre le colonne di Q, mentre i coefficienti utilizzati per costruirli a partire da A sono memorizzati nella porzione triangolare alta di R.

For $ii = 0, \ldots, n-1$, Do

(1) $R_{ii,ii:n-1} = A^T_{:,ii} * A_{:,ii:n-1}$

(2) $s = \sqrt{R_{ii,ii}}$

(3) $A(:, ii) = A(:, ii)/s$

(4) $R(ii, ii : n-1) = R(ii, ii : n-1)/s$

(5) $A(:, ii+1 : n-1) = A(:, ii+1 : n-1) - A(:, ii) * R(ii, ii+1 : n-1)$

End For

L'algoritmo di Gram-Schmidt seleziona la colonna ii di A, ne calcola il prodotto scalare con se stessa ($R(ii, ii)$) e con tutte le successive, memorizzando queste proiezioni sulla riga ii di R, dalla posizione ii fino a fine riga (**passo 1**). Successivamente scala la colonna ii di A in modo che abbia norma unitaria (**passo 3**), e scala la riga R del punto 1 per lo stesso fattore.

Nel **passo 5**, ad ogni colonna a destra della colonna selezionata viene sottratta la colonna in esame moltiplicata per i relativo coefficiente rappresentante la sua proiezione sulla colonna ii.

Il **passo 1** rappresenta il prodotto tra la trasposta di $A(:, ii)$ con la matrice $A(:, ii : n)$, è quindi il prodotto di una riga di lunghezza M con una matrice di dimensione $M \times N - ii - 1$. Il risultato è un vettore riga di lunghezza $N - ii - 1$.

Il **passo 3** è la scalatura di un vettore colonna, mentre il **passo 4** rappresenta la scalatura di una riga.

Il **passo 5** infine è un aggiornamento di una matrice di dimensione $M \times N - ii - 2$ utilizzando una matrice generata dal prodotto di una colonna di lunghezza M con una riga di lunghezza $N - ii - 2$ (update di rango 1).

Nell'implementazione del codice scalare, è utile produrre 3 funzioni:

- una funzione che calcola il prodotto $x^T B$, dove x è una colonna di A, mentre B individua una serie di colonne di A.
- una funzione che scala una riga/colonna per una costante.
- una funzione che aggiorna una matrice t.c. $B = B - xy$, dove B è una porzione di A, x è una vettore colonna, y un vettore riga.

Le funzioni suggerite possono operare direttamente sulle matrici A (input), e R (che verrà inizializzata a 0). Come suggerimento, utilizzare le definizioni di funzione contenute nel sorgente `res.c`.

L'implementazione CUDA dell'algoritmo utilizza lo stesso schema descritto, è necessario quindi fornire 3 kernel che implementino le 3 funzioni prncipali dell'algoritmo.

Per il kernel che implementa il **passo 1**, assegnare ad ogni thread il compito di calcolare un elemento del vettore riga: sarà quindi necessaria una griglia di dimensione sufficiente a calcolare $N - ii - 1$ elementi.

Per il kernel che implementa la scalatura (**passi 3 e 4**), ogni thread lanciato scala un singolo elemento. Attenzione: non è necessario passare come parametro al kernel il coefficiente di scalatura, ma solo il puntatore ad esso.

Per il terzo kernel infine, assegnare ad un blocco di threads il compito di aggiornare una intera riga. In questo caso utilizzare prima una griglia monodimensionale di dimensione (M, 1, 1), poi fornire una implementazione capace di sfruttare una griglia bidimensionale di dimensione (512, $(M + 511)/512$,1) in modo da poter gestire matrici più grandi.

Per tutti i kernel, utilizzare una dimensione di blocco fissa (512, 1, 1).

Inizializzare una matrice i cui elementi sulla diagonale siano:

$$A(ii, ii) = ii + 1 \qquad ii = 0, N - 1$$

mentre inizializzare i restanti elementi a zero.

Si testino matrici input 400×300 e 1000×800. Fornire per entrambi i casi, il tempo di cpu, il tempo di gpu e il tempo gpu + trasferimento dati, relativi speedup e bande di processamento (utilizzare il numero di elementi di input come dimensione dei dati).