Esercitazione 05

7 novembre 2023

Lo scopo di questa esercitazione è quello di implementare diversi varianti della moltiplicazione di matrici tenendo conto della località di memoria. In particolare si vedrò come moltiplicare due matrici $n \times n$, nel nostro caso di numeri floating point a precisione singola (`float`).

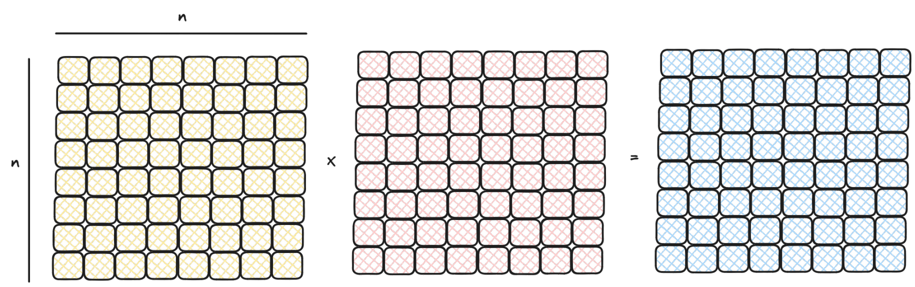

Figure 1: Moltiplicazione di matrici

Si assumo le matrici come rappresentate in modo contiguo in memoria (i.e., un singolo vettore) riga per riga (i.e., rappresentazione *row major*). Ne segue che l'elemento di posizione i, j di una matrice M di n righe n colonne sarà in posizione `M[i * n + j]`.

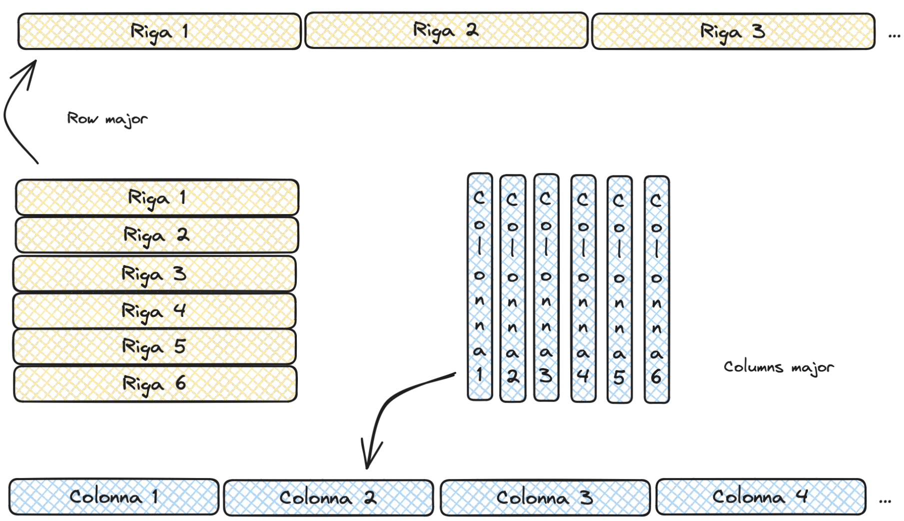

Figure 2: Rappresentazione per righe e per colonne

Le funzioni da implementare sono tre (di cui la prima è già implementata):

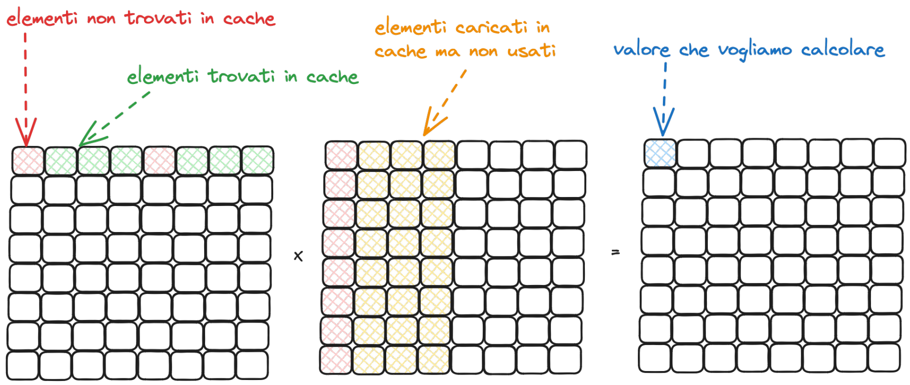

Figure 3: Effetto sulla cache dell'implementazione "naive" della moltiplicazione di matrici

1. Nel primo caso si deve implementare la semplice moltiplicazione righe per colonne. Questa è già implementata e fornisce una baseline per i successivi due metodi
2. Nel secondo caso si deve considerare il fatto che nella seconda matrice accediamo colonna per colonna, quindi in una rappresentazione row major abbiamo i valori delle righe che sono consecutivi in memoria, non quelli delle colonne. Si deve quindi cambiare la rappresentazione della seconda matrice in modo che sia in forma *column major* (si veda la figura), ovvero dove i dati sono memorizzati in modo che i dati delle colonne siano consecutivi. Si implementi poi la moltiplicazione righe per colonne di due matrici in cui la prima è in forma row major e la seconda in forma column major.
3. Infine si vuole implementare una moltiplicazione a blocchi. In questo caso le due matrici che vogliamo moltiplicare (chiamiamole A e B) sono divise in blocchi di dimensione fissata e possiamo riscrivere il risultato della moltiplicazione come somma di moltiplicazioni di questi blocchi. Formalmente:

   $$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

   Dove $C_{1,1}$ può venire calcolato come $A_{1,1} \times B_{1,1} + A_{1,1} \times B_{2,1} + A_{1,2} \times B_{1,1} + A_{1,2} \times B_{2,1}$. In pratica l'effetto sarebbe quello di avere una funzione per calcolare la moltiplicazione di due "blocchi" mettendo il risultato in C e iterando sulle matrici A e B blocco per blocco. Il vantaggio sarebbe quello che ciascuna di quelle moltiplicazioni può essere fatta su matrici abbastanza piccole da stare in cache. In pseudocodice questo prenderebbe la seguente forma:

   ```
   for i = 1 to n with step s1
   ```

```
    for j = 1 to n with step s2
        for k = 1 to n with step s3
            TMP =  matmul(A[i:i+s1, k:k+s3], B[k:k+s3, j:j+s2])
            C[i:i+s1, j:j+s2] = C[i:i+s1, j:j+s2] + TMP
```

questo significa che è utile implementare una funzione a cui passare le tre matrici A, B e C ma in cui viene chiesto di fare la moltiplicazione solo per un certo range di righe e colonne date dai valori di i, j, k come punto di partenza e dai tre parametri $s1$, $s2$ e $s3$. Una signature della funzione da implementare è presente nel file `matrix_multiply.c` con il nome `kernel`.

Suggerimenti

- Controllate che le moltiplicazioni di matrici che implementate siano corrette, viene fornito un metodo per stampare le matrici, provate a vedere che la moltiplicazione "naive" e quella che implementate voi corrispondano!
- Quando accedete a una matrice in forma column major ricordate che il calcolo dell'indice corretto è diverso da quello per la forma row major.
- Verificate come cambiano le prestazioni al variare di $s1$, $s2$ e $s3$ (i.e., la dimensione dei blocchi).