Linear discriminant analisys con CUDA C

Francesco Polvere *

12 novembre 2018

Sommario

In questo documento viene presentata una implementazione dell'algoritmo di linear discriminant analisys tramite l'utilizzo della GPU sulla piattaforma CUDA. L'implementazione parallela viene confrontata con l'implementazione classica in C, tramite la misura di tempo di esecuzione e colcolatone lo speed-up.

1 Introduzione

Da una analisi del metodo di LDA, si può notare la presenza di numerose operazioni su matrici che presentano un alto grado di parallelizzabilità.

2 Analisi dello stato dell'arte

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

*F. Polvere, Corso di GPU computing, A/A 2017-2018, Universitá degli studi di Milano, via Celoria 28, Milano, Italia
E-mail: francesco.polvere@studenti.unimi.it

3 Modello teorico

LDA è un metodo utilizzato per trovare una combinazione lineare di features che caratterizzato o separano due o più classi di oggetti o eventi. Il combinazione risultante può essere utilizzata come classificatore lineare o ancor prima della classificazione, come operazione di riduzione di dimensionalità.

3.1 LDA Multiclasse

Supponiamo di avere n classi. La Within-scatter matrix è calcolata come:

$$Sw = \sum_{i=1}^{C} \sum_{xappCi} (x - x_i)(x - x_i)'$$

Mentre la between-scatter matrix è calcolata come:

$$Sb = \sum_{i=1}^{n} m_i(\overline{x}_i - \overline{x})(\overline{x}_i - \overline{x})'$$

Dove $\overline{x}$ é la media totale di tutte le classi, m_i é il numero di dati di training per ogni classe.
Dopo aver ottenuto Sb e Sw, vogliamo trovare l'equazione lineare che massimizza l'equazione in figura.

$$J(W) = \frac{|W^T S_b W|}{|W^T S_w W|}$$

Si può dimostrare che la trasformazione W può essere ottenuta risolvendo un problema sugli autovalori:

$$S_b W = \lambda S_w W$$

3.2 Parallelizzazione di LDA utilizzando CUDA

Il dataset utilizzato è diviso in tre file, uno per ogni classe, contenente le osservazioni per ognuna di esse. Una volta caricati in memoria host, vengono copiati sulla sulla memoria device, in particolare nella global memory, il cui spazio è rilasciato solo al termine dell'esecuzione dell'intero programma.
L'allocazione su host è di tipo *pinned* per permettere l'esecuzione sovrapposta del caricamento da host a device di ogni classe ed il calcolo della media della classe stessa.

Assunzioni e notazioni Si assume che i dati siano stati estratti da una distribuzione gaussiana.
Il numero di osservazioni è uguale per ogni classe.

- C: numero di classi (o matrici)
- N: numero di osservazioni (o righe della matrice)
- M: numero di feature (o colonne della matrice)

Media della i-esima classe Per ogni classe vengono eseguite una *cudaMemcpyAsync* per portare i dati sulla global memory del device ed il kernel che calcola ne la media (figura 1).
I risultati sono mantenuti in global memory e non riportati in memoria host per ottimizzarne le prestazioni.

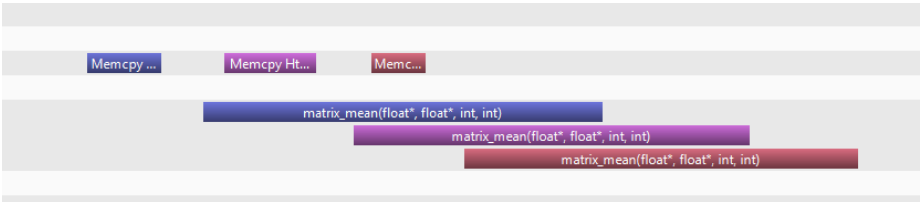

Figura 1: Gli stream per il calcolo della media

Media totale La media totale è calcolata con un kernel (...) che esegue la somma degli C-vettori risultati dal calcolo precedente e dividendo per uno scalare che è il numero di classi.
Questi n-vettori avranno hanno dimensione

Between-scatter matrix Il calcolo della Between-scatter matrix necessità di tre piccoli kernel.
Il kernel *diff_vect* si occupa di calcolare la differenza tra due vettori di dimensione M, nel caso specifico tra il vettore media locale i-esimo e media globale. Successivamente si utilizza il kernel *vector_prod* per il calcolo del prodotto tra il vettore in uscita dal kernel precedente e la sua trasposta. Il risultato è una matrice quadrata di dimensione M X M.
L'ultimo punto è la somma di tutte le C matrici in uscita dal kernel precedente. Tutte e tre le operazioni sono eseguite da tutte le classi in stream diversi per ogni classe per permettere un livello di esecuzione concorrente maggiore.

Within-scatter matrix Il calcolo della Within-scatter matrix

4 Simulazione ed esperimenti

Sono stati sviluppati due algoritmi. Quello sequenziale e quello parallelo.

4.1 Dataset

Per il benchmark é stato utilizzato il dataset Iris, introdotto da Ronald Fisher. Il dataset consiste in tre classi, ognuna delle quali rappresenta una specie di pianta, con un numero di istanze di 50 per ogni classe, per un totale di 150 istanze. Le variabili considerate sono quattro e consistono nella lunghezza e larghezza del sepalo e del petalo.

5 Risultati ottenuti

Il software è stato eseguito su un computer con le seguenti caratteristiche:

- GPU: NVIDIA GeForce 610M, compute capability 2.1, architettura FERMI (2.1) con 48 Cores/SM.
- CPU: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2401 Mhz, 4 core, 8 processori logici

Grafici

6 Conclusioni

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.