Esame di Programmazione su Architetture Parallele

Metodo del simplesso per la risoluzione della programmazione lineare

Belliato Riccardo (mat. 142652)   Simone Tomada

2022-08-26

Abstract

In questa relazione si propone una implementazione del metodo del simplesso a due fasi in CUDA per la risoluzione dei problemi di programmazione lineare in forma canonica.

Dopo una breve descrizione dell'algoritmo, seguirà la discussione su alcune scelte implementative.

Infine verranno valutate performance e scalabilità della soluzione proposta confrontando i tempi di esecuzione dell'algoritmo su istanze a dimensione crescente generate casualmente.

Contents

Introduzione al metodo del simplesso

Problemi di ottimizzazione lineare

Nell'ambito della Ricerca Operativa (Operations Research) uno dei principali argomenti è la cosiddetta **ottimizzazione lineare**, ossia lo studio di una classe di problemi del tipo:

$$\begin{gathered}\min / \max c^T x \\ \text{subject to} \\ Ax \lesseqgtr b\end{gathered}$$

con $x \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^{1 \times m}$.

In altre parole, si vogliono trovare dei valori per le componenti del vettore x (di dimensione n) tali da massimizzare (o minimizzare) il valore di una *funzione lineare* variabili (detta **funzione obiettivo**, mentre il vettore c è chiamato **vettore dei costi**), dati una serie di m vincoli espressi nel sistema di disequazioni lineari $Ax \lesseqgtr b$ (dove A è detta **matrice dei vincoli** e b **vettore dei termini noti**).

Questa classe di problemi permette di modellare un gran numero di situazioni reali in molteplici ambiti (ottimizzazione dei costi, creazione di orari, *etc.*), oltre che alcuni problemi NP-hard come il *Knapsack* o il *Vertex Cover*.

Dal punto di vista dell'algebra lineare, un problema in n varibili non è altro che uno spazio $\mathbb{R}^n$, la funzione obiettivo è una retta nello spazio, mentre i vincoli definiscono un *poliedro* nello spazio.

Utilizzando i teoremi e le tecniche dell'algebra lineare è stato possibile creare degli algoritmi per risolvere i problemi di ottimizzazione, come il **simplesso**, i quali si prestano molto bene ad essere parallelizzati (in quanto operano su matrici).

Problemi risolvibili, non risolvibili, illimitati

Dato un problema di ottimizzazione, questo può essere di tre tipi:

- ammissibile (*feasible*): ossia esistono uno o più vettori che moltiplicati per il vettore dei costi assegnano alla funzione obiettivo il valore massimo (o minimo possibile) e tutti i vincoli sono rispettati,
- non ammissibile (*infeasible*): se non esistono soluzioni
- illimitati (*unbounded*): se per una o più componenti della soluzione è possibile aumentarne (o ridurne) il valore all'infinito senza mai violare i vincoli
- degeneri (*degenerate*): sono problemi ammissibili in cui più soluzioni di base corrispondono allo stesso vertice del poliedro. In linea teorica possono far ciclare l'algoritmo (ossia l'algoritmo torna a controllare basi già viste in precedenza), ma all'atto pratico questo può essere evitato introducendo delle euristiche. Per semplicità la nostra implementazione una volta riconosciuto un problema come degenere lo segnala all'utente e termina senza cercare di risolverlo.

Forma canonica e forma standard

I problemi di massimizzazione possono essere convertiti in problemi di minimizzazione (e viceversa), così come è possibile manipolare le singole disequazioni. Queste operazioni servono a riportare i problemi in una forma che permetta di utilizzarli da parte dei solver. In particolare vengono utilizzate la forma *canonica* e la forma *standard*.

Un problema (di massimizzazione) è in forma canonica se è nella forma

$$\begin{gathered} \max c^T x \\ \text{subject to} \\ Ax \leq b \\ x \geq 0 \end{gathered}$$

Considerando che qualsiasi disequazione nella forma $\alpha x \leq y$ può essere convertita in una equazione equivalente $\alpha x + \delta = b$ definiamo la forma standard di un problema in forma canonica

$$\begin{gathered} \max (c|0)^T x \\ \text{subject to} \\ (A|I)x = b \\ x \geq 0 \end{gathered}$$

con $(A|I) \in \mathbb{R}^{m\times(n+m)}$, $(c|0) \in \mathbb{R}^{n+m}$ e I la matrice di identità di dimensione m.

In altre parole aggiungiamo una nuova variabile al problema (detta variabile *slack*) per ogni disequazione.

La forma standard è quella che viene utilizzata dagli algoritmi di soluzione.

Metodo del simplesso a due fasi

Il metodo del simplesso è un'algoritmo per risolvere i problemi di ottimizzazione lineari.

Si basa sul teorema secondo il quale le soluzioni (dette *soluzioni di base*) di un qualsiasi problema in forma standard sono i *vertici* del poliedro costruito sui vincoli.

Ogni vertice è individuato da una o più *soluzioni di base ammissibili*, cioè tutti i valori delle variabili in base sono maggiori o uguali a 0, mentre il valore delle altre variabili è 0. Se in base sono presenti uno o più valori uguali a 0, si dice che la base è *degenere*.

Esistono diverse implementazioni del metodo del simplesso, per questo progetto verrà implementato il cosiddetto *simplesso a due fasi con il metodo del tableau*

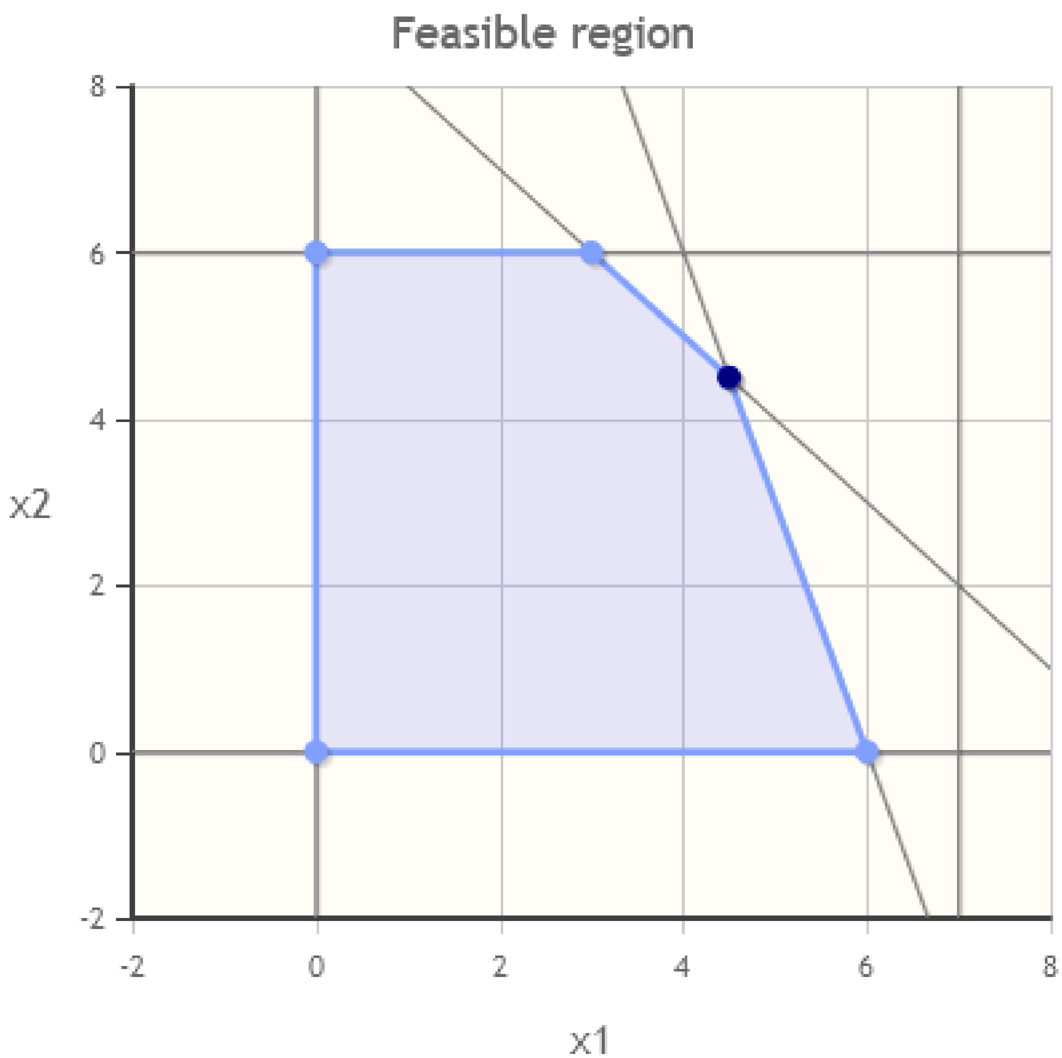

Figure 1: Esempio di problema con due variabili. Il vertice evidenziato in blu è la soluzione

Cos'è il tableau

Il tableau di un simplesso è la struttura dati su cui viene eseguito l'algoritmo del simplesso.

Si tratta di una matrice composta nel seguente modo:

Vettore dei costi

x_1	x_2	s_1	s_2	s_3	RHS
−3	−2	0	0	0	0
2	1	1	0	0	10
1	1	0	1	0	8
1	0	0	0	1	4

Valore funzione obiettivo

Matrice dei vincoli

Termini noti

Nella prima riga della matrice (R_0) è presente il vettore dei costi. Inizialmente il vettore dei costi contiene i coefficienti della funzione obiettivo con segno invertito. La base viene memorizzata in un vettore a parte.

Algoritmo di soluzione

Algorithm 1: Simplex

Data: $T \in \mathbb{R}^{(m+1)\times(n+1)}, \mathcal{B} \in \mathbb{N}^m$
Output: x come soluzione, $\mathcal{B}$ base della soluzione e $T[0][n]$ valore ottimo della funzione obiettivo
OPPURE problema UNBOUNDED
$A \leftarrow T[1..m][0..n-1]$; `/* La notazione` $[a..b]$ `indica una porzione di array dall'indice` a `all'indice` b `*/`
$b \leftarrow T[1..m][n]$;
$c \leftarrow T[0][0..n-1]$;
while $\exists i : c[i] < 0$ **do**
  Pick $h \in \{0, \dots, n-1\} : c[h] = \min_{i=0,\dots,n-1} c[i]$;
  if $\forall i \in \{0, \dots, m-1\} \Rightarrow A[0..m-1][h] \leq 0$ **then**
    return *problem UNBOUNDED*;
  end
  Pick $k \in \{0, \dots, m-1\} : c[k] = \min_{i=0,\dots,m-1} \left\{ \frac{b[i]}{A[i][h]} \wedge A[i][h] > 0 \right\}$;
  $p \leftarrow A[k][h]$;
  $\mathcal{B}[k] \leftarrow h$;
  $R_p \leftarrow copy(T[k][0..n])$;
  $C_p \leftarrow copy(T[0..m][h+1])$;
  for $i \leftarrow 0$ *to* m **do**
    for $j \leftarrow 0$ *to* n **do**
      if $i = k$ **then**
        $T[i][j] = \frac{1}{p}T[i][j]$;
      end
      else
        $T[i][j] = T[i][j] - \frac{C_p[i]}{p} R_p[j]$;
      end
    end
  end
end
$x[0..n-1] \leftarrow 0$;
for $i \leftarrow 0$ *to* $m-1$ **do**
  $x[\mathcal{B}[i]] \leftarrow b[i]$;
end
return $(x, \mathcal{B}, T[0][n])$; `/*` $T[0][n]$ `è il valore della funzione obiettivo */`

Complessità Considerando che l'algoritmo non fa altro che esplorare le soluzioni di base ammissibili, l'algoritmo esplora al massimo $O\binom{n+m}{m}$ (con n numero di variabili e m il numero di vincoli).

Nei casi peggiori il simplesso può raggiungere complessità esponenziali. Tuttavia nel corso degli anni si è visto EMPIRICAMENTE come questa complessità venga raggiunta in pochissimi casi specifici, anzi l'algoritmo si è rivelato molto efficiente su problemi "reali", dove in media raggiunge una complessità nell'ordine di $O(mn)$, anche a confronto con altri algoritmi di soluzione (polinomiali per costruzione) come il metodo dei punti interni.

Da queste considerazioni si può concludere anche come la complessità del simplesso dipenda molto dall'istanza del problema fornito in input.

Fase 1

In questa fase si costruisce il tableau e si verifica se il problema di partenza è ammissibile o meno. Per fare ciò si risolve il cosiddetto **problema ausiliario**

$$\begin{gathered}\max(0|-1)^T x \\ \text{subject to} \\ (A|I|I)x = b \\ x \geq 0\end{gathered}$$

In pratica aggiungo al problema originario m nuove variabili (dette **variabili artificiali**) e cerco di azzerarne la somma negata. Se ciò avviene il problema di partenza è ammissibile e la soluzione di base ottenuta può essere utilizzata come base di partenza per la fase 2 dell'algoritmo, altrimenti il problema originale non è ammissibile. Se nella base finale una delle variabili artificiali è rimasta in base il problema è degenere. Per lo pseudocodice si veda l'algoritmo 2.

Fase 2

Terminata la prima fase si eliminano le variabili artificiali dal tableau e si sostituisce la funzione obiettivo con quella del problema originale, in seguito si procede a risolvere il problema originale. Per lo pseudocodice si veda l'algoritmo 3.

Simplesso a due fasi

Si veda l'algoritmo 4

Scelte implementative

Schema di parallelizzazione

Per introdurre parallelismo si è deciso di parallelizzare le singole operazioni svolte sui vettori e sulle matrici, mantenendo intatta la struttura dell'algoritmo.

Così facendo non si prevede una riduzione della complessità teorica dell'algoritmo, ma ottenere un miglioramento empirico sui tempi necessari.

Gestione della memoria

Implementando gli algoritmi in CUDA, merita particolare risalto il modo in cui viene organizzata la memoria:

- i dati del problema originale sono memorizzati in memoria host, mentre il tableau viene costruito e memorizzato nella memoria globale della scheda video
- i dati del problema originale vengono memorizzati utilizzando memoria *page-locked*, questo per permettere i trasferimenti di memoria in parallelo con i CUDA Stream (e quindi ottimizzare la fase di costruzione del tableau in global memory)

Algorithm 2: Simplex-Phase1

```
Data: A ∈ ℝ^{m×n}, b ∈ ℝ^m in forma standard (senza var. slack)
Output: T ∈ ℝ^{(n+m+1)×(m+1)} tableau per la fase 2 e B ∈ ℕ^m base ammissibile OPPURE problema
        INFEASIBLE o DEGENERE
/* riempimento tableau                                                              */
T ∈ ℝ^{(m+1)×(n+2m+1)};
T[0][0..n+m−1] ← 0;
T[0][n+m..n+2m−1] ← 1;
T[0][n+2m] ← 0;
T[1..m][n+2m] ← b;
T[1..m][0..n−1] ← A;
T[1..m][n..n+m−1] ← I ;                                           /* variabili slack */
T[1..m][n+m..n+2m−1] ← I ;                                   /* variabili artificiali */
/* Nego eventuali disequazioni con termine noto negativo (in modo da rendere la base
   iniziale ammissibile)                                                            */
for i ← 1 to m do
    if T[i][n+2m] < 0 then
        for j ← 0 to n+2m do
            T[i][j] = −T[i][j];
        end
    end
end
B ← [n+m, ..., n+2m−1] ;              /* base iniziale con le variabili artificiali */
/* esprimo la funzione obiettivo solo in funzione delle variabili non di base       */
for i ← 1 to m do
    for j ← 0 to n+2m do
        T[0][j] ← T[0][j] − (T[0][B[i]] / T[i][B[i]]) T[i][j]
    end
end
(x, B, f) ← Simplex(T, B);
if f < 0 then
    return problem INFEASIBLE;
end
if ∃i : n+m ≤ B[i] < n+2m then
    return problem DEGENERE;
end
/* Elimino le variabili artificiali                                                 */
T[0..m][n+m] ← copy(T[0..m][n+2m]);
drop(T[0..m][n+m+1..n+2m]);
return (T, B)
```

Algorithm 3: Simplex-Phase2

Data: $T \in \mathbb{R}^{(m+1)\times(n+m+1)}, \mathcal{B} \in \mathbb{N}^m, c \in \mathbb{R}^n$
Output: soluzione al problema iniziale OPPURE problema UNBOUNDED
$T[0][0..n-1] \leftarrow -c$;
$T[0][n..n+m] \leftarrow 0$;
`/* esprimo la funzione obiettivo solo in funzione delle variabili non di base */`
for $i \leftarrow 1$ *to* m **do**
  for $j \leftarrow 0$ *to* $n+m$ **do**
    $T[0][j] \leftarrow T[0][j] - \frac{T[0][\mathcal{B}[i]]}{T[i][\mathcal{B}[i]]} T[i][j]$
  end
end
return $Simplex(T, \mathcal{B})$

Algorithm 4: 2Phases-Simplex

Data: $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$
Output: soluzione al problema OPPURE problema INFEASIBLE o UNBOUNDED o DEGENERE
$(T, \mathcal{B}) \leftarrow Simplex-Phase1(A, b)$;
return $Simplex-Phase2(T, \mathcal{B}, c)$

- il vettore della base è memorizzato come memoria *mapped page-locked*, in modo da potervi accedere sia lato host che lato GPU in qualsiasi momento senza dover gestire i trasferimenti
- siccome l'algoritmo opera spesso su singole righe o colonne è fondamentale memorizzare i dati in memoria globale in modo da minimizzare il numero di accessi strided o disallineati. Per questo motivi si è deciso il seguente schema di memorizzazione (visibile in figura 2):
  - la matrice del tableau viene allocata con *cudaMallocPitch* ed è memorizzata in maniera *lineare* per **colonne**. In altre parole l'implementazione utilizza il tableau trasposto. L'algoritmo infatti accede più spesso ai vettori delle variabili che ai vettori dei vincoli. Per l'accesso ai singoli vincoli (in questo caso le colonne della matrice memorizzata in global memory) si è visto che l'approccio migliore (dal punto di vista del tempo necessario ad eseguire l'operazione) è quello di utilizzare comunque gli accessi strided.
  - il vettore dei termini noti non viene memorizzato in fondo al tableau, ma nella prima riga della matrice. In questo modo nel passaggio dalla fase 1 alla fase 2 non è necessario spostare la colonna nella matrice, ma è sufficiente troncare il numero di righe senza dover intervenire sull'allocazione della memoria.
  - il vettore dei costi non viene memorizzato nella matrice del tableau, ma in un vettore a parte e il valore della funzione obiettivo è nel primo elemento di questo vettore (analogalmente a quanto fatto con il vettore dei termini noti al punto precedente).

Risultati sperimentali

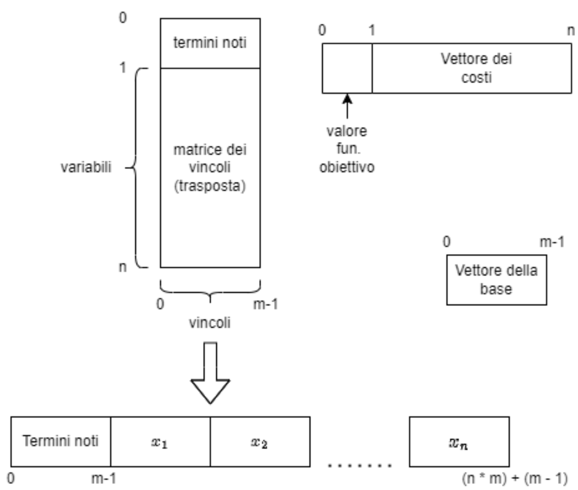

Figure 2: Schema di memorizzazione in memoria globale