Implementazione di un modulo per il calcolo del determinante in una libreria ibrida per l'Algebra Lineare.

Antonio Miti

July 7, 2014

Progetto iniziale

Concetti chiave:

$[A]_{\text{device}}$ $[A]_{\text{host}}$

- 2 copie della matrice, una memorizzata sull'host e l'altra su device. (Memorizzate come array 1d di *float*.)

Progetto iniziale

Concetti chiave:

$$[A]_{\text{device}} \Downarrow [A']_{\text{device}} \qquad [A]_{\text{host}} \Downarrow [A']_{\text{host}}$$

- 2 copie della matrice, una memorizzata sull'host e l'altra su device. (Memorizzate come array 1d di *float*.)
- 2 copie di ciascun metodo.

Progetto iniziale

Concetti chiave:

$$
\begin{array}{ll}
[A]_{\text{device}} & [A]_{\text{host}} \\
\Downarrow & \Downarrow \\
[A']_{\text{device}} & [A']_{\text{host}} \\
\Downarrow & \Downarrow \\
\vdots & \vdots \\
\Downarrow & \Downarrow \\
[A'']_{\text{device}} & [A'']_{\text{host}}
\end{array}
$$

- 2 copie della matrice, una memorizzata sull'host e l'altra su device. (Memorizzate come array 1d di *float*.)
- 2 copie di ciascun metodo.
- Possibilita' di effettuare calcoli complessi separatamente sul solo host o sul device.

Progetto iniziale

Concetti chiave:

$$
\begin{array}{ccc}
[A]_{\text{device}} & & [A]_{\text{host}} \\
\Downarrow & & \Downarrow \\
[A']_{\text{device}} & & [A']_{\text{host}} \\
\Downarrow & & \Downarrow \\
\vdots & & \vdots \\
\Downarrow & & \Downarrow \\
[A'']_{\text{device}} & \xleftrightarrow{\text{sync}} & [A'']_{\text{host}}
\end{array}
$$

- 2 copie della matrice, una memorizzata sull'host e l'altra su device. (Memorizzate come array 1d di *float*.)
- 2 copie di ciascun metodo.
- Possibilita' di effettuare calcoli complessi separatamente sul solo host o sul device.
- Metodi per sincronizzare le 2 copie della matrice. (Per minimizzare il trasferimento dei dati fra host e device.)

Progetto iniziale

il codice

- L'oggetto principale della libreria è costituito dalla classe *matrice*.

```
class matrice{
float *A_host ; // puntatore alle variabli host
float *A_dev ;  //puntatori alle variabli device
unsigned int ROW,COL,N;

bool flagCuda; // flag presenza di scheda cuda
bool flagMemory; // flag di sincronizzazione copia host -device
bool flagSquare;  //flag matrice quadrata
[...]
```

Progetto iniziale

il codice

- L'oggetto principale della libreria è costituito dalla classe *matrice*.

```
class matrice{
float *A_host ; // puntatore alle variabli host
float *A_dev ;  //puntatori alle variabli device
unsigned int ROW,COL,N;

bool flagCuda; // flag presenza di scheda cuda
bool flagMemory; // flag di sincronizzazione copia host -device
bool flagSquare;  //flag matrice quadrata
[...]
```

- Il sorgente di ogni modulo è salvato in header file distinti.

Progetto iniziale

il codice

- L'oggetto principale della libreria è costituito dalla classe *matrice*.

```
class matrice{
float *A_host ; // puntatore alle variabli host
float *A_dev ;  //puntatori alle variabli device
unsigned int ROW,COL,N;

bool flagCuda; // flag presenza di scheda cuda
bool flagMemory; // flag di sincronizzazione copia host -device
bool flagSquare;  //flag matrice quadrata
[...]
```

- Il sorgente di ogni modulo è salvato in header file distinti.
- Anche i singoli CUDA kernel utilizzati sono salvati in specifici header.

Progetto iniziale

il codice

- L'oggetto principale della libreria è costituito dalla classe *matrice*.

```
class matrice{
float *A_host ; // puntatore alle variabli host
float *A_dev ;  //puntatori alle variabli device
unsigned int ROW,COL,N;

bool flagCuda; // flag presenza di scheda cuda
bool flagMemory; // flag di sincronizzazione copia host -device
bool flagSquare;  //flag matrice quadrata
[...]
```

- Il sorgente di ogni modulo è salvato in header file distinti.
- Anche i singoli CUDA kernel utilizzati sono salvati in specifici header.
- ... il modulo che verrà analizzato sarà quello di *"condensazione"*.

Algoritmi di condensazione

L'idea

Classe di algoritmi pensata esplicitamente per il calcolo dei determinanti. Basata sulla tesi:

$$\exists \mathsf{T} : \ \mathsf{Mat}(n,n) \to \ \mathsf{Mat}(n1, n-1)$$

tale che

$$\forall A \in \ \mathsf{Mat}(n,n) \quad \exists P_{(A)} \in \mathbb{K} \qquad | \quad \mathsf{det}(A) = P_{(A)} \cdot \mathsf{det}\big(\mathsf{T}(A)\big)$$

Algoritmi di condensazione

L'idea

Classe di algoritmi pensata esplicitamente per il calcolo dei determinanti. Basata sulla tesi:

$$\exists \mathsf{T} : \ \text{Mat}(n, n) \to \ \text{Mat}(n1, n-1)$$

tale che

$$\forall A \in \ \text{Mat}(n, n) \quad \exists P_{(A)} \in \mathbb{K} \qquad | \quad \det(A) = P_{(A)} \cdot \det\big(\mathsf{T}(A)\big)$$

$B = \mathsf{T}(A)$ è detta *condensazione* della matrice.

Algoritmi di condensazione

L'idea

L'applicazione consecutiva di tale operatore ...

$$\cdots \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

L'applicazione consecutiva di tale operatore ...

$$\cdots \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \boxed{\mapsto} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$\boxed{P_{(A_{n-3})}}$$

Algoritmi di condensazione

L'idea

L'applicazione consecutiva di tale operatore ...

$$
\begin{tikzcd}
\cdots \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} & \mapsto \arrow[d] & \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} & \mapsto \arrow[dl] & \begin{bmatrix} \boxed{D} \end{bmatrix} \\
 & P_{(A_{n-3})} & P_{(A_{n-2})} & &
\end{tikzcd}
$$

Algoritmi di condensazione

L'idea

L'applicazione consecutiva di tale operatore ...

$$\cdots \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \mapsto \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} \mapsto \begin{bmatrix} D \end{bmatrix}$$

$$P_{(A_{n-3})} \qquad P_{(A_{n-2})} \qquad D$$

Algoritmi di condensazione
L'idea

L'applicazione consecutiva di tale operatore ...

$$\cdots \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \mapsto \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} \mapsto \begin{bmatrix} D \end{bmatrix}$$

$$\text{det}(A_1) = \cdots P_{(A_{n-3})} \cdot P_{(A_{n-2})} \cdot D$$

Permette di calcolare il determinante della matrice come prodotto dei *Pivot*.

Algoritmi di condensazione

La formula di Salem-Said

$$\begin{pmatrix} \cdots & a_{0j} & \cdots & \boxed{a_{0l}} & \cdots \\ & \vdots & & \vdots & \\ \cdots & a_{ij} & \cdots & a_{il} & \\ & \vdots & & \vdots & \ddots \end{pmatrix}$$

$$\Downarrow \mathsf{T}_{\mathsf{SS}}$$

$$\begin{pmatrix} \ddots & \vdots & \\ & b_{i-1,j} & \\ & \vdots & \ddots \end{pmatrix}$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (primo elemento non nullo nella prima riga)

Algoritmi di condensazione

La formula di Salem-Said

$$\begin{pmatrix} \boxed{\cdots \quad a_{0j} \quad \cdots \quad \color{red}{a_{0l}} \quad \cdots} \\ \vdots \qquad \vdots \\ \cdots \quad \color{blue}{a_{ij}} \quad \cdots \quad a_{il} \\ \vdots \qquad \vdots \qquad \ddots \end{pmatrix}$$

$$\Downarrow \mathsf{T}_{\mathsf{SS}}$$

$$\begin{pmatrix} \ddots & \vdots & \\ & b_{i-1,j} & \\ & \vdots & \ddots \end{pmatrix}$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (primo elemento non nullo nella prima riga)
2. Scorro gli elementi. (nella matrici privata della riga e colonna del pivot)

Algoritmi di condensazione

La formula di Salem-Said

$$
\begin{pmatrix}
\cdots & \boxed{a_{0j}} & \cdots & \boxed{a_{0l}} & \cdots \\
 & \vdots & & \vdots & \\
\cdots & \boxed{a_{ij}} & \cdots & \boxed{a_{il}} & \\
 & \vdots & & \vdots & \ddots
\end{pmatrix}
$$

$$\Downarrow \mathsf{T}_{\mathsf{SS}}$$

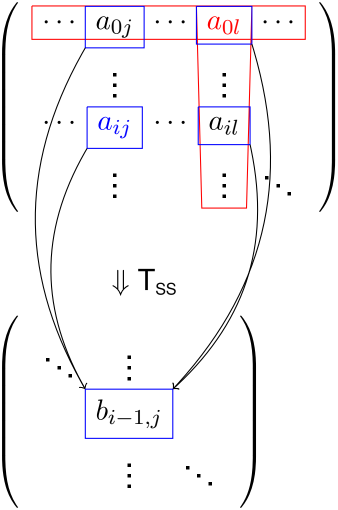
$$
\begin{pmatrix}
\ddots & \vdots & \\
 & b_{i-1,j} & \\
 & \vdots & \ddots
\end{pmatrix}
$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (primo elemento non nullo nella prima riga)
2. Scorro gli elementi. (nella matrici privata della riga e colonna del pivot)
3. Calcolo determinante 2x2 evidenziato.

Algoritmi di condensazione

La formula di Salem-Said

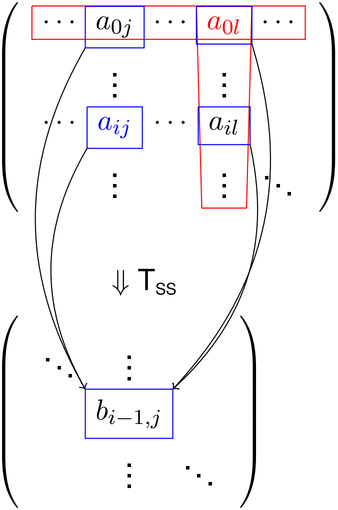


Routine di singola condensazione:

1. Scelta elemento Pivot. (primo elemento non nullo nella prima riga)
2. Scorro gli elementi. (nella matrici privata della riga e colonna del pivot)
3. Calcolo determinante 2x2 evidenziato.
4. Elemento della matrice trasformata = determinante appena calcolato

Algoritmi di condensazione

La formula di Salem-Said



Routine di singola condensazione:

1. Scelta elemento Pivot. (primo elemento non nullo nella prima riga)
2. Scorro gli elementi. (nella matrici privata della riga e colonna del pivot)
3. Calcolo determinante 2x2 evidenziato.
4. Elemento della matrice trasformata = determinante appena calcolato

Risulta la seguente espressione per $P_{(A)} = \dfrac{1}{(a_{0l})^{n-2}}$

Algoritmi di condensazione
Versione modificata

$$\begin{array}{c} \begin{array}{ccccc} a_{0,0} & \cdots & a_{0,l} & \cdots & a_{0,n-1} \end{array} \\ \left(\begin{array}{ccccc} \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots \\ a_{n-1,0} & \cdots & \boxed{a_{n-1,l}} & \cdots & a_{n-1,n-1} \end{array}\right) \end{array}$$

$$\Downarrow \mathrm{T}_{\mathrm{mod}}$$

$$\left(\begin{array}{ccc} \ddots & \vdots & \\ & b_{i,j} & \\ & \vdots & \ddots \end{array}\right)$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (Elemento in modulo maggiore nell'ultima riga)

Algoritmi di condensazione

Versione modificata

$$\begin{array}{ccccc} a_{0,0} & \cdots & a_{0,l} & \cdots & a_{0,n-1} \end{array}$$

$$\begin{pmatrix} \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots \\ \frac{a_{n-1,0}}{a_{n-1,l}} & \cdots & \boxed{1} & \cdots & \frac{a_{n-1,n-1}}{a_{n-1,l}} \end{pmatrix}$$

$$\Downarrow \mathrm{T_{mod}}$$

$$\begin{pmatrix} \ddots & \vdots & \\ & b_{i,j} & \\ & \vdots & \ddots \end{pmatrix}$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (Elemento in modulo maggiore nell'ultima riga)
2. Divisione ultima riga per il valore del Pivot.

Algoritmi di condensazione

Versione modificata

$$
\begin{array}{c}
\begin{array}{ccccc} a_{0,0} & \cdots & a_{0,n-1} & \cdots & a_{0,l} \end{array} \\
\begin{pmatrix}
\vdots & & \vdots & & \vdots \\
\vdots & & \vdots & & \vdots \\
\frac{a_{n-1,0}}{a_{n-1,l}} & \cdots & \frac{a_{n-1,n-1}}{a_{n-1,l}} & \cdots & \boxed{1}
\end{pmatrix}
\end{array}
$$

$$\Downarrow \mathrm{T}_{\mathrm{mod}}$$

$$
\begin{pmatrix}
\ddots & \vdots & \\
 & b_{i,j} & \\
 & \vdots & \ddots
\end{pmatrix}
$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (Elemento in modulo maggiore nell'ultima riga)
2. Divisione ultima riga per il valore del Pivot.
3. Scambio colonna del pivot con ultima riga.

Algoritmi di condensazione
Versione modificata

$$\begin{pmatrix} & \vdots & \vdots \\ \cdots & a'_{i,j} & a'_{i,n-1} \\ & \vdots & \vdots \\ \cdots & a'_{n-1,i} \quad \cdots & \boxed{1} \end{pmatrix}$$

$$\Downarrow \mathrm{T}_{\mathrm{mod}}$$

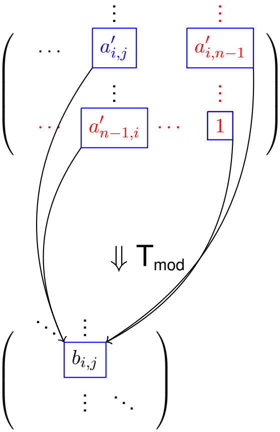
$$\begin{pmatrix} \ddots & \vdots & \\ & b_{i,j} & \\ & \vdots & \ddots \end{pmatrix}$$

Routine di singola condensazione:

1. Scelta elemento Pivot. (Elemento in modulo maggiore nell'ultima riga)
2. Divisione ultima riga per il valore del Pivot.
3. Scambio colonna del pivot con ultima riga.
4. Selezione elemento della matrice fin qui trasformata (privata della riga e colonna del pivot).

Algoritmi di condensazione

Versione modificata



Routine di singola condensazione:

1. Scelta elemento Pivot. (Elemento in modulo maggiore nell'ultima riga)
2. Divisione ultima riga per il valore del Pivot.
3. Scambio colonna del pivot con ultima riga.
4. Selezione elemento della matrice fin qui trasformata (privata della riga e colonna del pivot).
5. Calcolo dell'elemento condensato come determinante 2x2.

Algoritmi di condensazione

Vantaggi

- In questo caso il valore $P_{(A)}$, necessario per ricostruire il determinante, coincide con il valore del pivot: $P_{(A)} = a_{n-1,l}$.
  (in quanto viene condensata la matrice A' di pivot pari ad 1 tramite la formula di Salem Said.)

Algoritmi di condensazione

Vantaggi

- In questo caso il valore $P_{(A)}$, necessario per ricostruire il determinante, coincide con il valore del pivot: $P_{(A)} = a_{n-1,l}$.
  (in quanto viene condensata la matrice A' di pivot pari ad 1 tramite la formula di Salem Said.)
- A differenza dell'algoritmo precendente, non c'è rischio di over-flow o under-flow durante il susseguirsi degli step di condensazione.

Algoritmi di condensazione

Vantaggi

- In questo caso il valore $P_{(A)}$, necessario per ricostruire il determinante, coincide con il valore del pivot: $P_{(A)} = a_{n-1,l}$.
  (in quanto viene condensata la matrice A' di pivot pari ad 1 tramite la formula di Salem Said.)
- A differenza dell'algoritmo precendente, non c'è rischio di over-flow o under-flow durante il susseguirsi degli step di condensazione.

Sia $S \simeq < |a_*| >$ la taglia della matrice di partenza e S' della matrice condensata. Si ha:

$$S'_{\text{SS}} \simeq S^2$$

$$S'_{\text{mod}} \simeq S + \alpha S \simeq S$$

(dove α è una quantità di modulo minore di 1).

Algoritmi di condensazione

Vantaggi

- In questo caso il valore $P_{(A)}$, necessario per ricostruire il determinante, coincide con il valore del pivot: $P_{(A)} = a_{n-1,l}$.
  (in quanto viene condensata la matrice A' di pivot pari ad 1 tramite la formula di Salem Said.)
- A differenza dell'algoritmo precendente, non c'è rischio di over-flow o under-flow durante il susseguirsi degli step di condensazione.

Sia $S \simeq < |a_*| >$ la taglia della matrice di partenza e S' della matrice condensata. Si ha:

$$S'_{\mathsf{SS}} \simeq S^2$$

$$S'_{\mathsf{mod}} \simeq S + \alpha S \simeq S$$

(dove α è una quantità di modulo minore di 1).

Attenzione!

Il determinante è definito come sommatoria di permetuzioni di molti elementi! Per calcolare la produttoria dei pivot sarà comunque necessario implementare una classe per gestire numeri di grande esponente.

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

1. Ricerca del pivot.

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

1. Ricerca del pivot.
2. Divisione della riga.

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

1. Ricerca del pivot.
2. Divisione della riga.
3. Scambio dell'ultima colonna.

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

1. Ricerca del pivot.
2. Divisione della riga.
3. Scambio dell'ultima colonna.
4. Condensazione vera e propria.

```
void matrice::Cpu_Step_Condensation_Simple(){
 int newROW = ROW-1; int newCOL = COL-1;

 for(int i=0 ; i<newROW ; i++)for(int j=0 ; j<newCOL ; j++)
  A_host[i*newCOL+j] =A_host[i*COL+j]-A_host[i*COL+COL-1]*A_host[(ROW
      -1)*COL+j];

 ROW=newROW; COL=newCOL;
[...]
```

Implementazione CPU

Per realizzare il singolo passo di condensazione sono stati realizzati quattro metodi. (nel file *"Condensation.cuh"*).

1. Ricerca del pivot.
2. Divisione della riga.
3. Scambio dell'ultima colonna.
4. Condensazione vera e propria.

```
void matrice::Cpu_Step_Condensation_Simple(){
 int newROW = ROW-1; int newCOL = COL-1;

 for(int i=0 ; i<newROW ; i++)for(int j=0 ; j<newCOL ; j++)
  A_host[i*newCOL+j] =A_host[i*COL+j]-A_host[i*COL+COL-1]*A_host[(ROW
      -1)*COL+j];

 ROW=newROW; COL=newCOL;
[...]
```

I quattro vengono accorpati un un quinto metodo che li cicla fino a condensare una data matrice ad un singolo elemento.

Implementazione CPU

Analisi

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.

Implementazione CPU

Analisi

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.

Implementazione CPU

Analisi

$$\begin{pmatrix} \boxed{a} & b & c & \boxed{d} \\ e & f & g & h \\ i & l & m & n \\ \boxed{o} & p & q & r \end{pmatrix}$$

Lettura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} \boxed{a} & b & c & \boxed{d} \\ e & f & g & h \\ i & l & m & n \\ \boxed{o} & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU
Analisi

$$\begin{pmatrix} \color{green}{a'} & b & c & \boxed{d} \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU
Analisi

$$\begin{pmatrix} a' & b' & c & d \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura
Scrittura

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU
Analisi

$$\begin{pmatrix} a' & b' & c' & d \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura ☐
Scrittura ☐

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & \boxed{d'} \\ \boxed{e} & f & g & \boxed{h} \\ i & l & m & n \\ \boxed{o} & p & q & r \end{pmatrix}$$

Lettura □

Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & f & g & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & \boxed{f'} & \boxed{g} & \boxed{h} \\ i & l & m & n \\ o & p & \boxed{q} & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & f' & g' & h \\ i & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & f' & g' & \boxed{h'} \\ i & \boxed{l} & m & \boxed{n} \\ o & \boxed{p} & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & f' & g' & h' \\ i' & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura $\square$
Scrittura $\square$

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione CPU

Analisi

$$\begin{pmatrix} a' & b' & c' & d' \\ e' & f' & g' & h' \\ i' & l & m & n \\ o & p & q & r \end{pmatrix}$$

Lettura □
Scrittura □

L'implementazione si avvantaggia dell'algoritmo modificato in 2 modi:

1. La matrice è memorizzata srotolata, c'è un pattern naturale secondo cui scorrere gli elementi.
2. La scelta di aver spostato i pivot nella cornice in basso a destra assicura di poter scrivere gli elementi della matrice condensanta sulla memoria riservata alla matrice iniziale senza incorrere in errori.

Implementazione GPU

- Come per il caso Cpu vengono realizzati 4 metodi che eseguono le operazioni chiave.

Implementazione GPU

- Come per il caso Cpu vengono realizzati 4 metodi che eseguono le operazioni chiave.
- Novità: ogni metodo chiama uno specifico kernel per manipolare la copia gpu della matrice.

Implementazione GPU

- Come per il caso Cpu vengono realizzati 4 metodi che eseguono le operazioni chiave.
- Novità: ogni metodo chiama uno specifico kernel per manipolare la copia gpu della matrice.
- Il metodo che realizza il ciclo complessivo ha una struttura ibrida. (Ad ogni step il valore del pivot viene memorizzato in un array sull'host) Questo ha 2 pregi:

Implementazione GPU

- Come per il caso Cpu vengono realizzati 4 metodi che eseguono le operazioni chiave.
- Novità: ogni metodo chiama uno specifico kernel per manipolare la copia gpu della matrice.
- Il metodo che realizza il ciclo complessivo ha una struttura ibrida. (Ad ogni step il valore del pivot viene memorizzato in un array sull'host) Questo ha 2 pregi:

1. Per il calcolo della produttoria la cpu è più adatta a manipolare dati in strutture personalizzate (la *struct "big number"*).

Implementazione GPU

- Come per il caso Cpu vengono realizzati 4 metodi che eseguono le operazioni chiave.
- Novità: ogni metodo chiama uno specifico kernel per manipolare la copia gpu della matrice.
- Il metodo che realizza il ciclo complessivo ha una struttura ibrida. (Ad ogni step il valore del pivot viene memorizzato in un array sull'host) Questo ha 2 pregi:

1. Per il calcolo della produttoria la cpu è più adatta a manipolare dati in strutture personalizzate (la *struct "big number"*).
2. Nella fase di divisione dell'ultima riga per il valore del pivot è comodo passare al kernel il valore del divisore come argomento. (si evita la congestione dei thread mentre tentano di accedere simultaneamente in lettura sulla stessa cella di memoria)

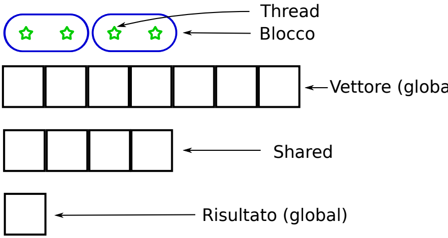
Implementazione GPU

Kernel ricerca del massimo:

Implementazione GPU

Kernel ricerca del massimo:

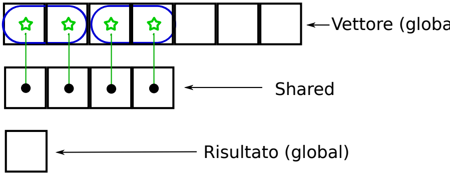
(si supponga una griglia 1D composta da 2 blocchi di 2 thread agenti su una lista di 7 numeri)



Implementazione GPU

Kernel ricerca del massimo:

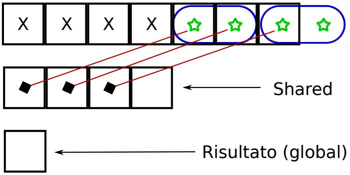
- I thread salvano i primi valori sulla shared



Implementazione GPU

Kernel ricerca del massimo:

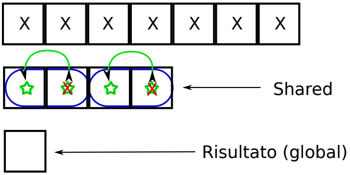
- I thread salvano i primi valori sulla shared
- Riduzione. (Ogni thread confronta sequenzialmente il valore appena salvato con il corrispettivo dopo N_{thread})



Implementazione GPU

Kernel ricerca del massimo:

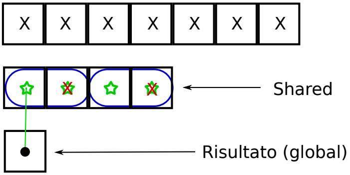
- I thread salvano i primi valori sulla shared
- Riduzione. (Ogni thread confronta sequenzialmente il valore appena salvato con il corrispettivo dopo N_{thread})
- Confronto telescopico. (Solo metà dei thread del blocco lavorano, confrontano il loro valore con quello della seconda metà)



Implementazione GPU

Kernel ricerca del massimo:

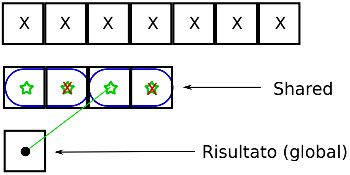
- I thread salvano i primi valori sulla shared
- Riduzione. (Ogni thread confronta sequenzialmente il valore appena salvato con il corrispettivo dopo N_{thread})
- Confronto telescopico. (Solo metà dei thread del blocco lavorano, confrontano il loro valore con quello della seconda metà)
- Confronto *atomico*. (I primi thread di ogni blocco contengono il massimo relativo del blocco. Procedono a confronatre tale valore con quello precaricato uno per volta regolati da un *lock*.)



Implementazione GPU

Kernel ricerca del massimo:

- I thread salvano i primi valori sulla shared
- Riduzione. (Ogni thread confronta sequenzialmente il valore appena salvato con il corrispettivo dopo N_{thread})
- Confronto telescopico. (Solo metà dei thread del blocco lavorano, confrontano il loro valore con quello della seconda metà)
- Confronto *atomico*. (I primi thread di ogni blocco contengono il massimo relativo del blocco. Procedono a confronatre tale valore con quello precaricato uno per volta regolati da un *lock*.)

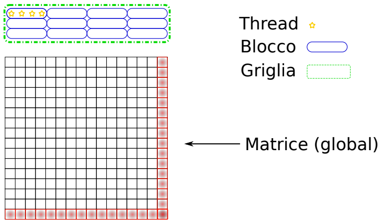


Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

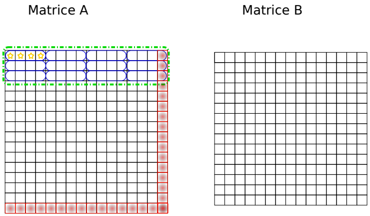


Implementazione GPU

Kernel passo di condensazione

Azione del kernel:

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

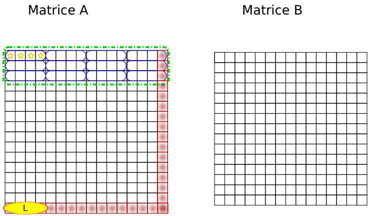


Implementazione GPU
Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del "pivot" orizzontale.



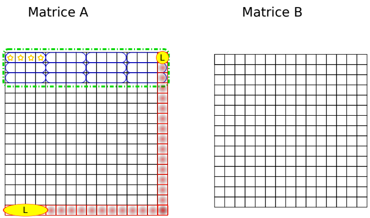
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del ”pivot” orizzontale.
2. Copia del ”pivot” verticale.



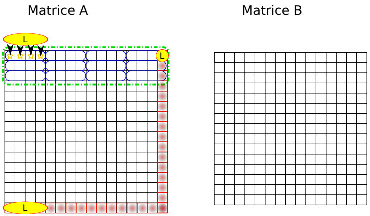
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del "pivot" orizzontale.
2. Copia del "pivot" verticale.
3. Copia dell'entry da condensare.



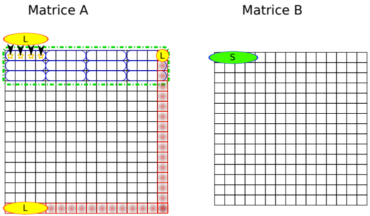
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del ”pivot” orizzontale.
2. Copia del ”pivot” verticale.
3. Copia dell’entry da condensare.
4. Scrittura del valore condensato.



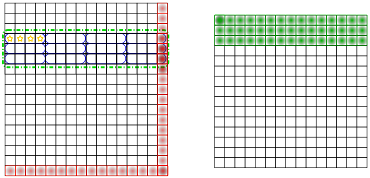
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del "pivot" orizzontale.
2. Copia del "pivot" verticale.
3. Copia dell'entry da condensare.
4. Scrittura del valore condensato.



Si ripete l'operazione "spostando" la griglia in verticale.

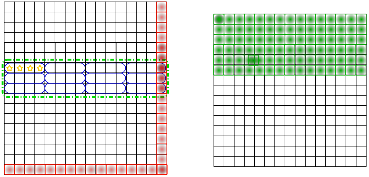
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del ”pivot” orizzontale.
2. Copia del ”pivot” verticale.
3. Copia dell’entry da condensare.
4. Scrittura del valore condensato.



Si ripete l’operazione ”spostando” la griglia in verticale.

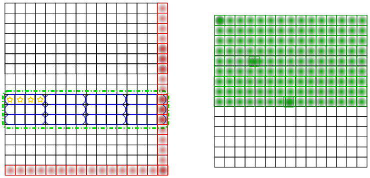
Implementazione GPU

Kernel passo di condensazione

(si supponga una griglia 2D composta da 4 blocchi 1D composti da 4 thread agenti su una matrice 16×16)

Azione del kernel:

1. Copia del ”pivot” orizzontale.
2. Copia del ”pivot” verticale.
3. Copia dell’entry da condensare.
4. Scrittura del valore condensato.



Si ripete l’operazione ”spostando” la griglia in verticale.

Implementazione GPU

Kernel passo di condensazione

Osservazioni:

Implementazione GPU

Kernel passo di condensazione

Osservazioni:

- Invocazione di una griglia di thread sufficentemente grande da poter ricoprire almeno una volta una riga della matrice completata. (
  in questo modo non è necessario far scorrere la griglia anche verso destra).

Implementazione GPU

Kernel passo di condensazione

Osservazioni:

- Invocazione di una griglia di thread sufficentemente grande da poter ricoprire almeno una volta una riga della matrice completata. (in questo modo non è necessario far scorrere la griglia anche verso destra).
- Nei passi 1,3,4 i thread contigui agiscono su celle di memorie contigue: Se la dimensione dei blocchi è proporzionale al *warp* è preservata la **coalescenza degli indirizzi**.

Implementazione GPU

Kernel passo di condensazione

Osservazioni:

- Invocazione di una griglia di thread sufficentemente grande da poter ricoprire almeno una volta una riga della matrice completata. (in questo modo non è necessario far scorrere la griglia anche verso destra).
- Nei passi 1,3,4 i thread contigui agiscono su celle di memorie contigue: Se la dimensione dei blocchi è proporzionale al *warp* è preservata la **coalescenza degli indirizzi.**
- Nel passo 2 è evidente un rischio di congestione in lettura di tutti i thread relativi ad una riga della griglia. E' utile sfruttare la **memoria texture.**

Implementazione GPU

Kernel passo di condensazione

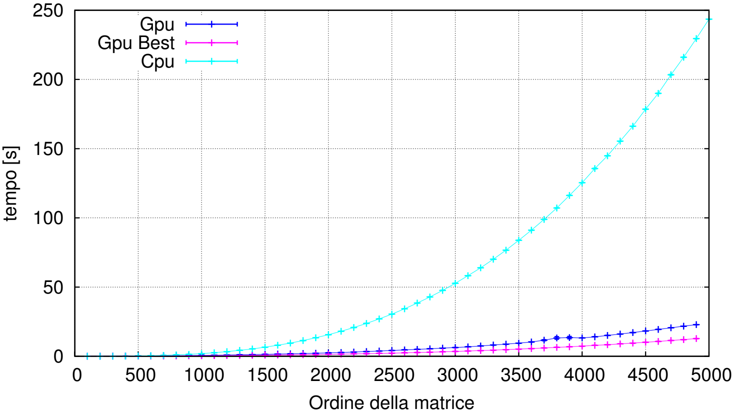
Osservazioni:

- Invocazione di una griglia di thread sufficentemente grande da poter ricoprire almeno una volta una riga della matrice completa. (in questo modo non è necessario far scorrere la griglia anche verso destra).
- Nei passi 1,3,4 i thread contigui agiscono su celle di memorie contigue: Se la dimensione dei blocchi è proporzionale al *warp* è preservata la **coalescenza degli indirizzi**.
- Nel passo 2 è evidente un rischio di congestione in lettura di tutti i thread relativi ad una riga della griglia. E' utile sfruttare la **memoria texture.**

Nella cartella *KernelTest/* sono presenti vari kernel per testare diverse strategie di utilizzo della memoria.

Conclusioni

Confronto tempo d'esecuzione vs Taglia della matrice



Conclusioni

Confronto tempo d'esecuzione vs Taglia della matrice (zoom)