

UNIVERSITÀ DEGLI STUDI DI FIRENZE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTO FINALE DEL CORSO DI PARALLEL COMPUTING:

Kernel Image Processing

Studente: Kevin Maggi

Anno Accademico 2021/2022

Overview

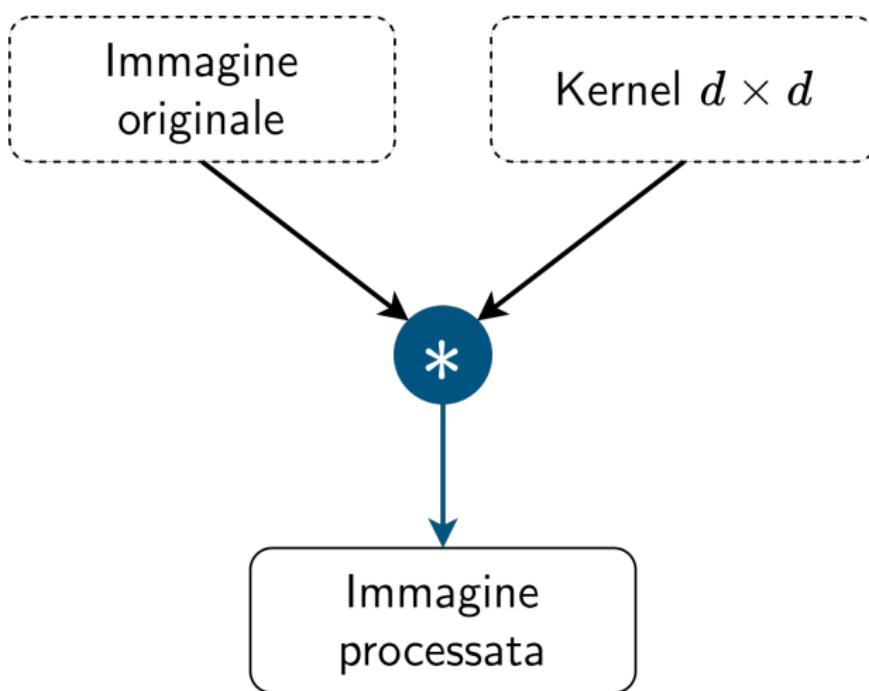
Contesto: Kernel Image Processing

Implementazioni: C sequenziale
C + OpenMP
CUDA-C

Obiettivo: Speed up

Kernel Image Processing

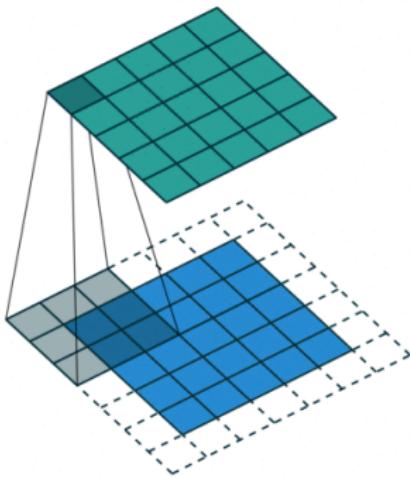
Principio



Kernel Image Processing

Convoluzione

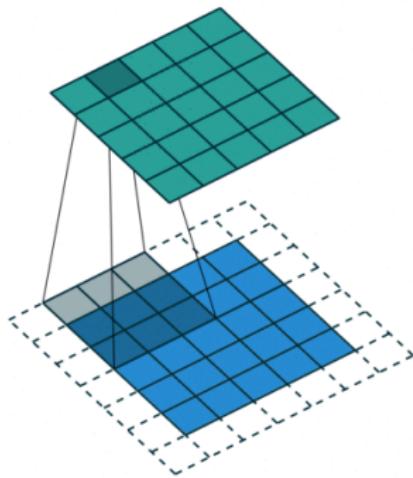
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

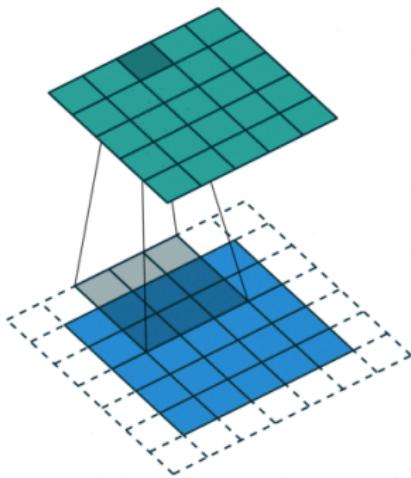
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

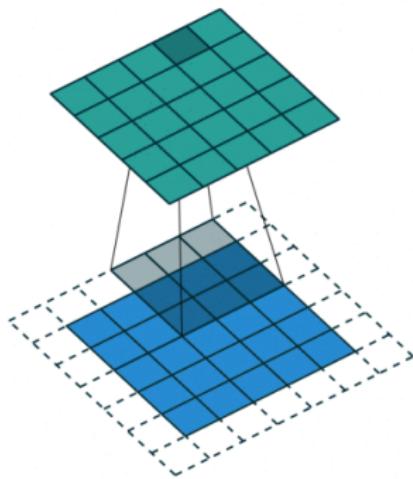
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

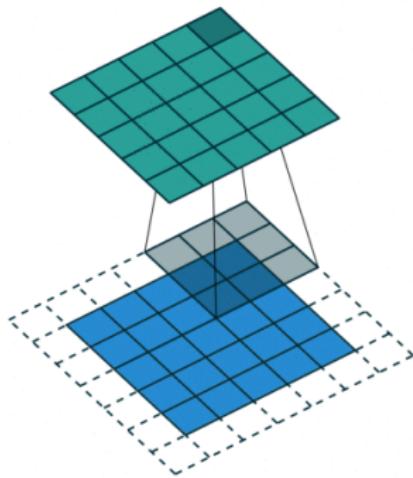
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

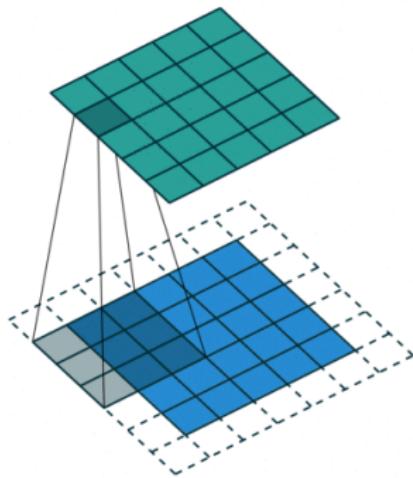
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

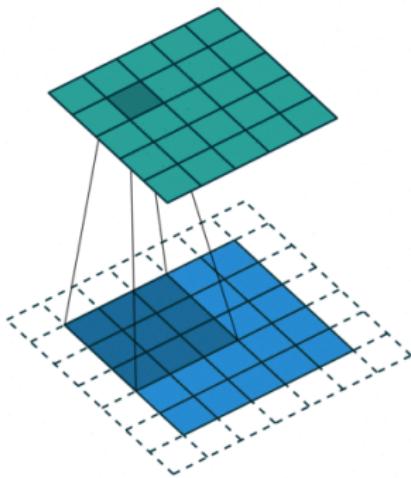
$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Convoluzione

$$P_{x,y} = \sum_{dx=-d}^d \sum_{dy=-d}^d K_{dx,dy} I_{x+dx, y+dy}$$



Kernel Image Processing

Edge Handling

Gestione degli elementi del bordo (di spessore $\frac{d-1}{2}$):

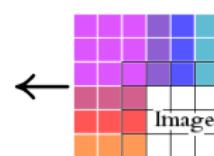
- Kernel crop
- Crop
- Mirror
- Wrap
- Extend

Kernel Image Processing

Edge Handling

Gestione degli elementi del bordo (di spessore $\frac{d-1}{2}$):

- Kernel crop
- Crop
- Mirror
- Wrap
- Extend



Blur



Blur

Box blur



$$K = \frac{1}{d^2} \underbrace{\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}}_d$$

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



$$K = \frac{1}{2^{2(d-1)}} \omega \omega^T$$

con ω d -esima riga del triangolo di Tartaglia

Pseudocodice

```
for y from 1 to height
    for x from 1 to width
        for c in channels
            sum = 0;
            for i from 1 to d
                for j from 1 to d

                    // edge handling

                    sum += I(x,y,c)*K(i,j);
            sum = sum/K.weight;
            P(x,y,c) = sum;
```

Profiling

Unico hotspot



100% del tempo di computazione

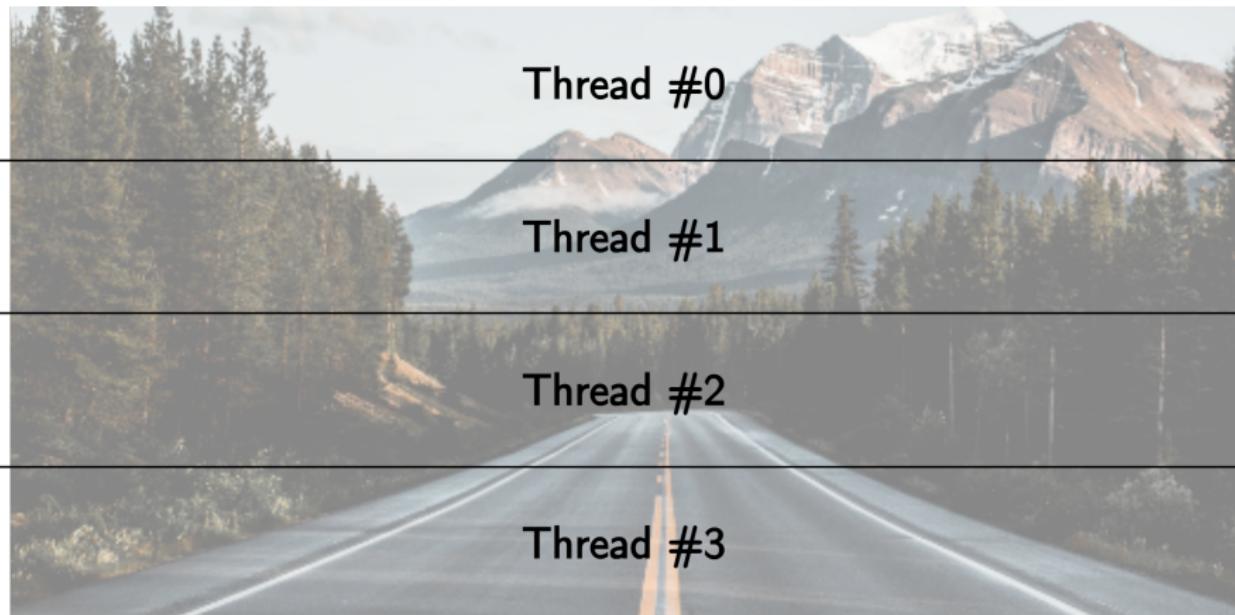
OpenMP

Stesso codice → una singola direttiva per il ciclo più esterno

```
#pragma omp parallel for
    default (none)
    shared(I, K, P)
    private(sum, indexes)
    firstprivate(radius)

    schedule(static)
    num_threads(4)
```

Decomposizione dei dati



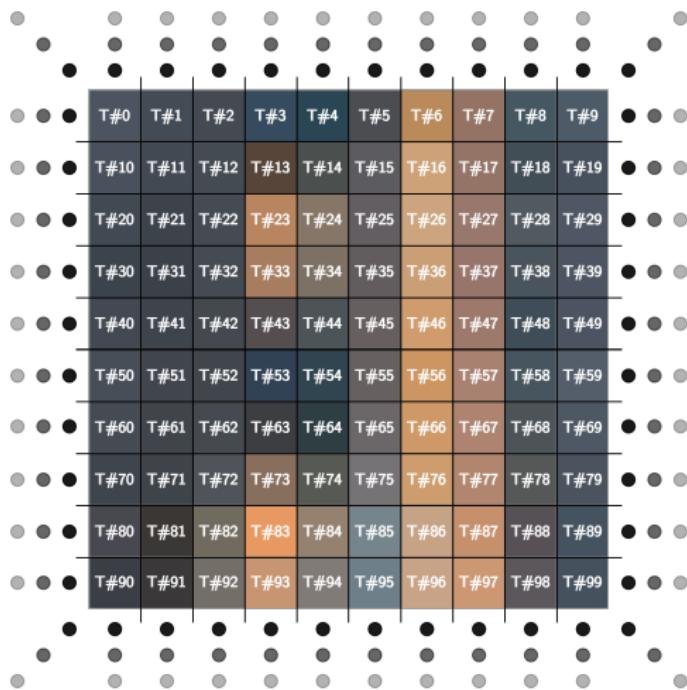
OpenMP

Alternative

```
#pragma parallel
for y from 1 to height
    for x from 1 to width
        for c in channels
            sum = 0;
            #pragma for reduction(+:sum)
            for i from 1 to d
                for j from 1 to d
                    sum += I(x,y,c)*K(i,j);
```

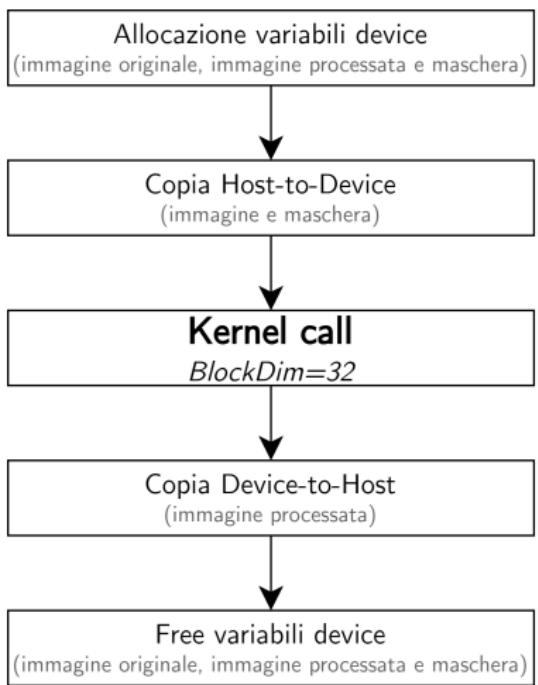
⇒ performance peggiori

Decomposizione dei dati



1° step

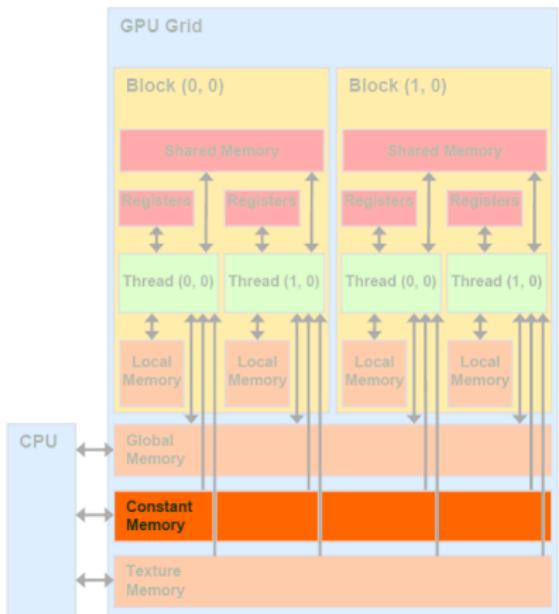
Versione naïve



```
x = fx(blockIdx.x, threadIdx.x);  
y = fy(blockIdx.y, threadIdx.y);  
  
for c in channels  
    sum = 0;  
    for i from 1 to d  
        for j from 1 to d  
  
            // edge handling  
  
            sum += I(x,y,c)*K(i,j);  
    sum = sum/K.weight;  
    P(x,y,c) = sum;
```

2^o step

Constant memory



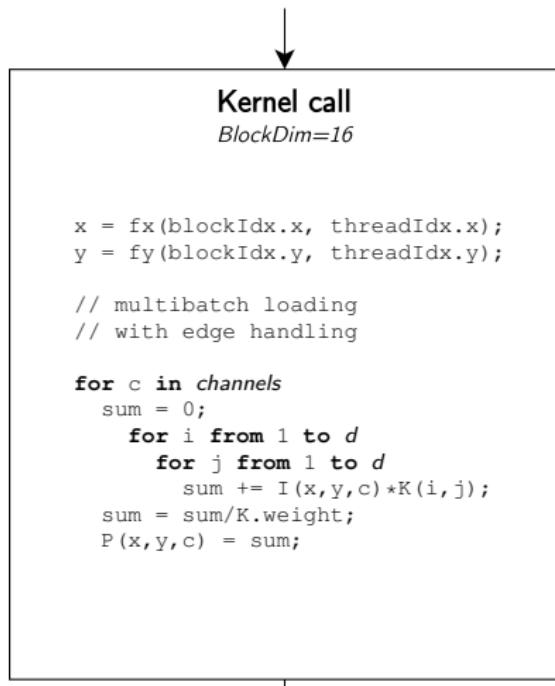
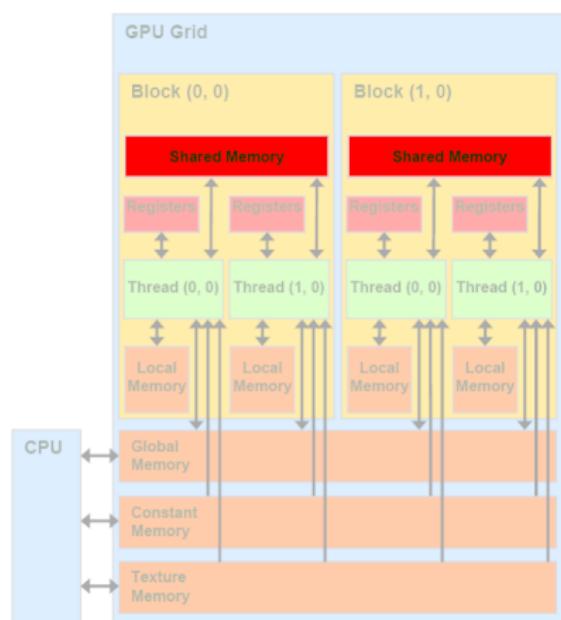
Allocazione statica



`__constant__ KERNEL[25*25]`

3^o step

Shared memory



4^o step

Pinned memory

$d = 25 \rightarrow \sim 150\text{MB}$ di memoria



con poca memoria può essere fatto swap su disco



immagine e maschera allocate con **pinned memory**

Test

Macchina:

- GPU: NVIDIA GeForce 940MX: Compute Capability 5.0, 512 CUDA core, Maxwell Architecture
- CPU: Intel Core i7-6500U con **Hyperthreading**: 2.5GHz up to 3.1GHz, 2 core/4 thread
- RAM: 8GB

Input:

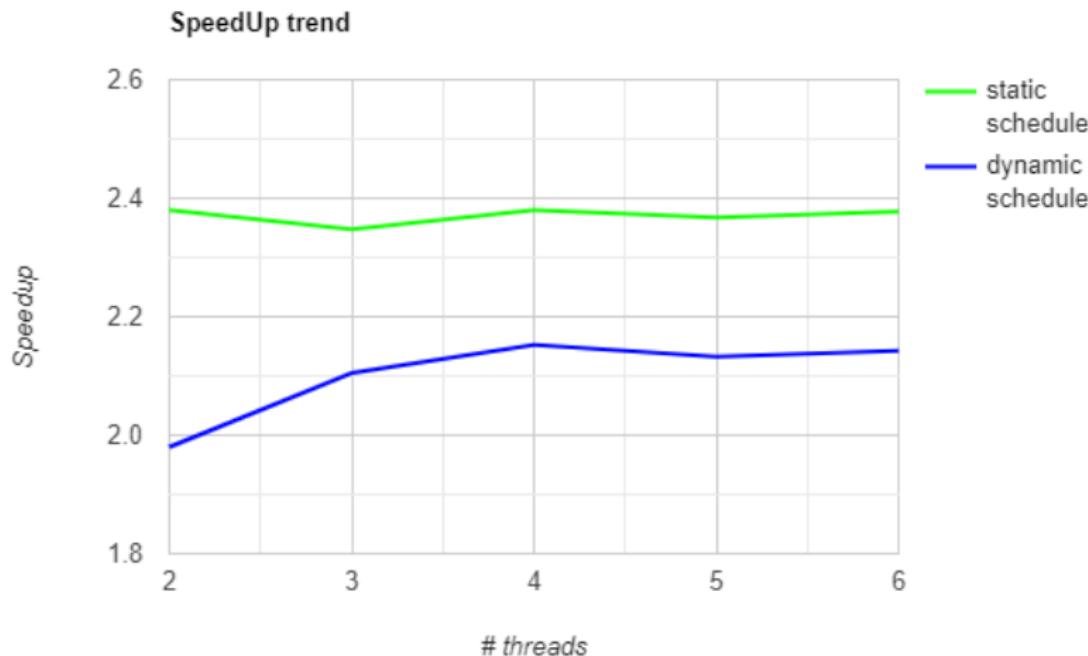
- immagini: 5K, 6K, 7K, 8K
- d : 7, 13, 19, 25

Risultati

OpenMP: scheduling statico, 4 thread

Input	speed up
5K	$d = 7$ 2.28
	$d = 13$ 2.32
	$d = 19$ 2.55
	$d = 25$ 2.38
6K	$d = 7$ 2.28
	$d = 13$ 2.31
	$d = 19$ 2.55
	$d = 25$ 2.38
7K	$d = 7$ 2.27
	$d = 13$ 2.32
	$d = 19$ 2.55
	$d = 25$ 2.39
8K	$d = 7$ 2.19
	$d = 13$ 2.12
Media	2.36

thread ottimale e scheduling migliore

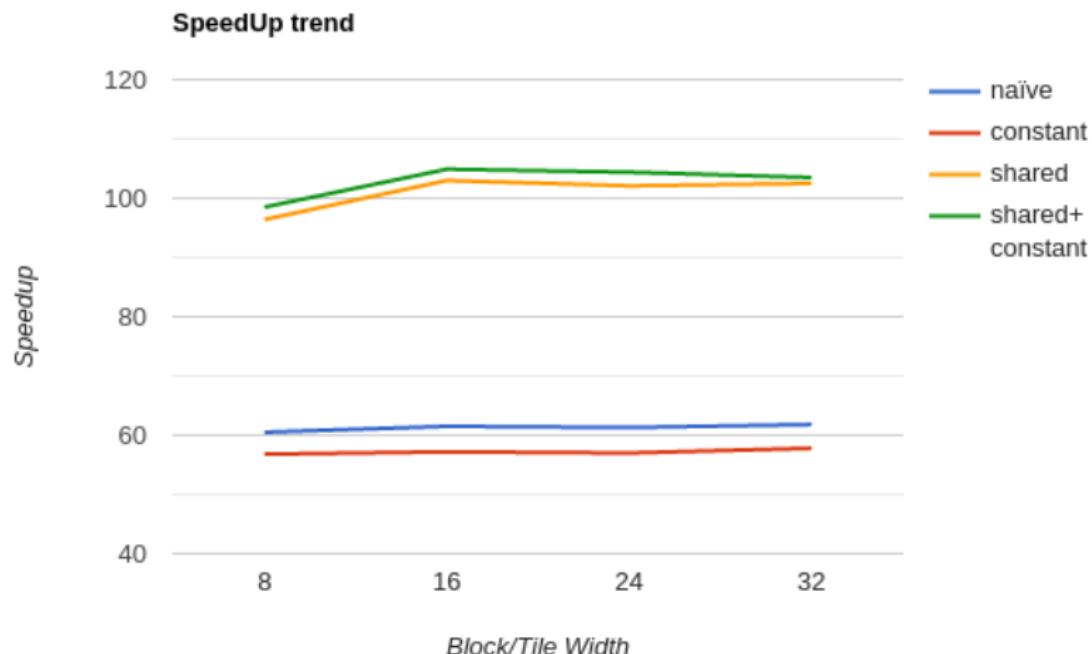


Risultati

CUDA: block width 32 / 16 (shared)

Input	speed up				
	naïve	constant	shared	constant+shared	
5K	$d = 7$	40.7	40.7	45.6	49.6
	$d = 13$	52.7	49.9	73.8	78.5
	$d = 19$	60.5	57.6	97.4	100.8
	$d = 25$	61.1	57.7	101.7	105.5
6K	$d = 7$	41.4	40.4	47.7	52.0
	$d = 13$	52.6	49.9	73.6	78.5
	$d = 19$	60.9	58.1	98.0	101.0
	$d = 25$	61.8	57.8	102.5	105.3
7K	$d = 7$	42.2	41.6	48.4	52.2
	$d = 13$	52.9	50.8	73.8	79.2
	$d = 19$	61.1	58.2	97.7	101.2
	$d = 25$	62.0	57.9	102.9	106.2
8K	$d = 7$	40.8	40.3	46.8	50.9
	$d = 13$	48.5	46.4	68.1	72.6
Media		53.0	50.6	77.4	81.4

Block width ottimale



Pinned memory profiling

Performance non migliori



↑ throughput (da 1.56 GB/s a 1.59 GB/s)



Istanziazione più lenta



Grazie per l'attenzione