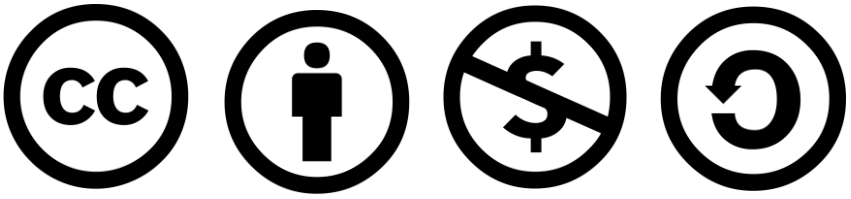


Report Common Assignment CUDA01

Counting Sort Algorithm

Lecturer: Francesco Moscato - fmoscato@unisa.it

Group:

- **Battipaglia Lucia** **0622701758** **l.battipaglia6@studenti.unisa.it**
- **Canzolino Gianluca** **0622701806** **g.canzolino3@studenti.unisa.it**
- **Farina Luigi** **0622701754** **l.farina16@studenti.unisa.it**

Sommario

Problem description

Lo scopo di questo studio è quello di esaminare le performance della parallelizzazione dell'algoritmo di ordinamento **"counting sort",** il quale è semplice da implementare senza il processo di parallelizzazione. Per raggiungere l'obiettivo, il codice è stato parallelizzato su una scheda video Nvidia attraverso la CUDA API. L'obiettivo è quello di misurare le performance offerte dall'algoritmo. Il kernel è stato modificato per operare su memorie diverse in modo tale da poter analizzare tutte le casistiche per valutarne l'efficienza.
Sono state utilizzate le seguenti combinazioni di memorie:

- *Global Memory*,
- *Global Memory e Shared Memory*,
- *Texture Memory e Global Memory*,
- *Texture Memory e Shared Memory*.

Experimental setup

Hardware

CPU

Per poter ottenere le informazioni riguardante la CPU, è stato usato il seguente comando:

```
!cat /proc/cpuinfo
```

Configuration 1

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 79
model name  : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping    : 0
microcode   : 0x1
cpu MHz     : 2199.998
cache size  : 56320 KB
physical id : 0
siblings    : 2
core id     : 0
cpu cores   : 1
apicid      : 0
initial apicid   : 0
fpu         : yes
fpu_exception    : yes
cpuid level : 13
wp          : yes
```

```
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd
ibrs ibpb stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
rtm rdseed adx smap xsaveopt arat md_clear arch_capabilities
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs taa
bogomips    : 4399.99
clflush size     : 64
cache_alignment  : 64
address sizes    : 46 bits physical, 48 bits virtual
power management:

processor   : 1
vendor_id   : GenuineIntel
cpu family  : 6
model       : 79
model name  : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping    : 0
microcode   : 0x1
cpu MHz           : 2199.998
cache size  : 56320 KB
physical id : 0
siblings    : 2
core id     : 0
cpu cores   : 1
apicid      : 1
initial apicid   : 1
fpu         : yes
fpu_exception    : yes
cpuid level : 13
wp          : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd
ibrs ibpb stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
rtm rdseed adx smap xsaveopt arat md_clear arch_capabilities
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs taa
bogomips    : 4399.99
clflush size     : 64
cache_alignment  : 64
address sizes    : 46 bits physical, 48 bits virtual
power management:
```

GPU

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
Sun Jan 23 15:06:21 2022

Device name: Tesla K80
Compute capability: 3.7

Clock Rate: 823500 kHz
Total SMs: 13
Shared Memory Per SM: 114688 bytes
Registers Per SM: 131072 32-bit
Max threads per SM: 2048
L2 Cache Size: 1572864 bytes
Total Global Memory: 11996954624 bytes
Memory Clock Rate: 2505000 kHz

Max threads per block: 1024
Max threads in X-dimension of block: 1024
Max threads in Y-dimension of block: 1024
Max threads in Z-dimension of block: 64

Max blocks in X-dimension of grid: 2147483647
Max blocks in Y-dimension of grid: 65535
Max blocks in Z-dimension of grid: 65535

Shared Memory Per Block: 49152 bytes
Registers Per Block: 65536 32-bit
Warp size: 32
```

GPU Bandwidth

```
Device 0: Tesla K80
 Range Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)    Bandwidth(MB/s)
   1000                      202.8
   101000                   5941.6
   201000                   6660.5
   301000                   6858.3
   401000                   6896.0
   501000                   7181.2
   601000                   7223.8
   701000                   7261.4
   801000                   7433.8
```

```
  901000            7398.1

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)    Bandwidth(MB/s)
  1000                      409.0
  101000            6495.7
  201000            7137.9
  301000            7317.1
  401000            7426.9
  501000            7486.8
  601000            7552.9
  701000            7571.9
  801000            7613.0
  901000            7622.8

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)    Bandwidth(MB/s)
  1000                      253.4
  101000           25891.7
  201000           41917.7
  301000           60030.0
  401000           73698.0
  501000           80673.8
  601000           91201.4
  701000           99054.6
  801000           90735.4
  901000           77983.9
```

Software

Google Colab

Performance

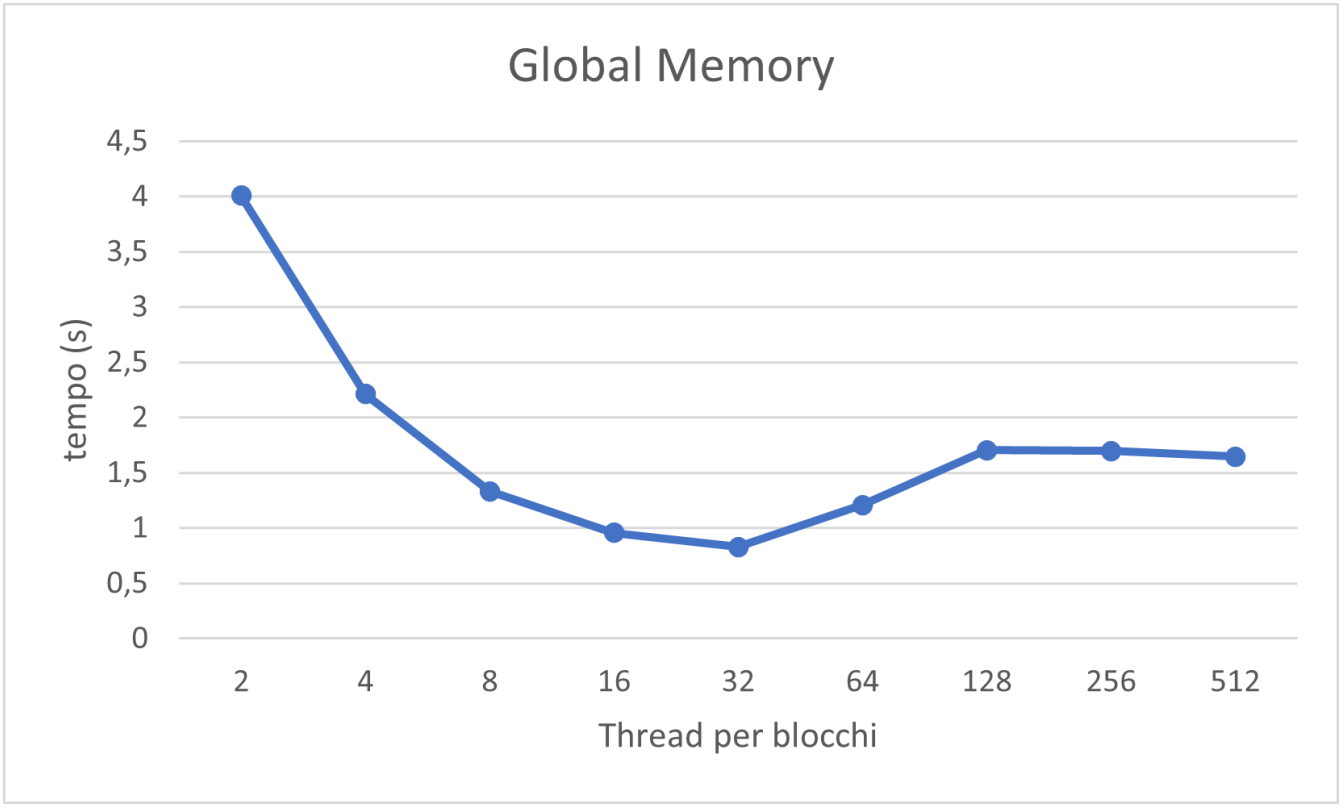
Le performance sono state valutate con diverse dimensioni di blocchi e threads:

- 512 threads → in questo caso abbiamo 2048/512=4 blocchi per SM. L'occupazione è del 100%.
- 256 threads → in questo caso abbiamo 2048/256=8 blocchi per SM. L'occupazione è del 100%.
- 128 threads → in questo caso abbiamo 2048/128=16 blocchi per SM. L'occupazione è del 100%.
- 64 threads → in questo caso abbiamo 2048/64=32 blocchi per SM. L'occupazione è del 100%.
- 32 threads → in questo caso abbiamo 2048/32=64 blocchi per SM. L'occupazione è del 100%.
- 16 threads → in questo caso abbiamo 2048/16=128 blocchi per SM. L'occupazione è del 100%.
- 8 threads → in questo caso abbiamo 2048/8=256 blocchi per SM. L'occupazione è del 100%.
- 4 threads → in questo caso abbiamo 2048/4=512 blocchi per SM. L'occupazione è del 100%.
- 2 threads → in questo caso abbiamo 2048/2=1024 blocchi per SM. L'occupazione è del 100%.

Case study n°1 – Global Memory - Global Memory

In questo caso di studio, è stato analizzato l'algoritmo di Counting Sort utilizzando esclusivamente la memoria globale; in particolare il vettore da ordinare è stato copiato all'interno della memoria globale, il conteggio delle occorrenze e la riduzione somma vengono effettuate usando la memoria globale.

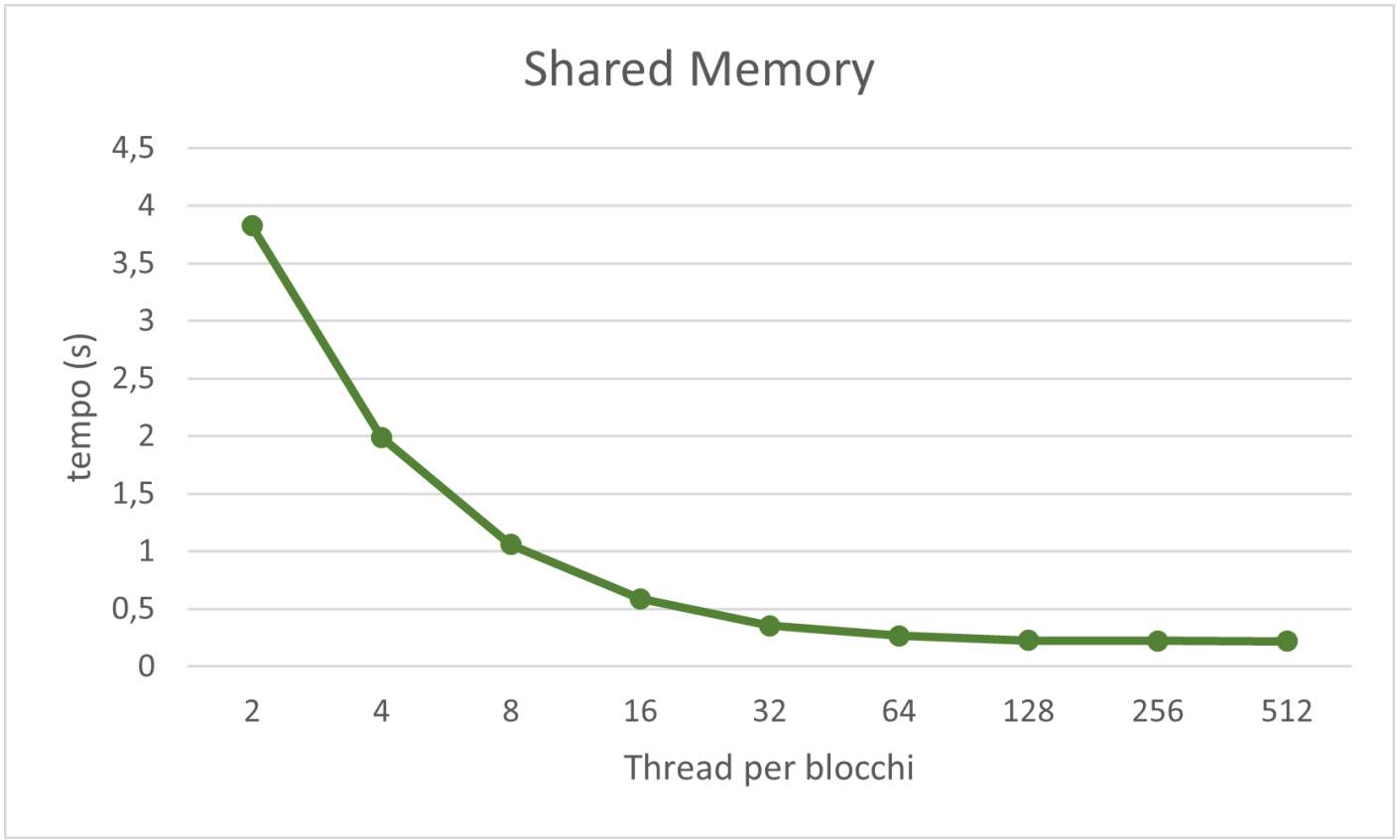
Size	*BlockSize*	*GridSize*	*Time*
33554432	*2*	*13312*	*4,008162*
33554432	*4*	*6656*	*2,21111*
33554432	*8*	*3328*	*1,33296*
33554432	*16*	*1664*	*0,956022*
33554432	*32*	*832*	*0,830413*
33554432	*64*	*416*	*1,207627*
33554432	*128*	*208*	*1,706399*
33554432	*256*	*104*	*1,696734*
33554432	*512*	*52*	*1,645474*



Case study n°2 – Global Memory - Shared Memory

In questo caso di studio, è stato analizzato l'algoritmo di Counting Sort utilizzando la memoria globale e la memoria shared; in particolare il vettore da ordinare è stato copiato all'interno della memoria globale, il conteggio delle occorrenze e la riduzione somma vengono effettuate usando la memoria shared.

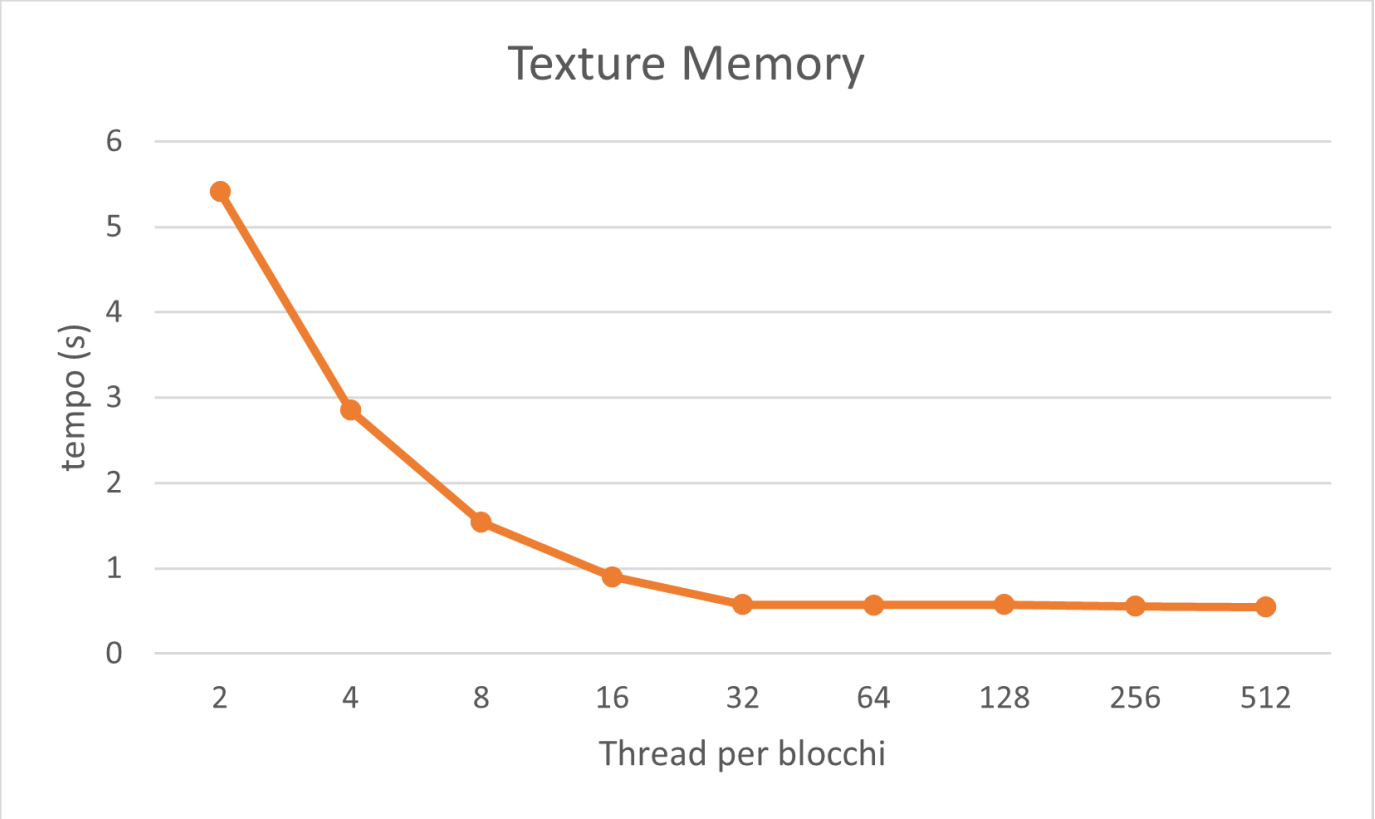
Size	*BlockSize*	*GridSize*	*Time*
33554432	*2*	*13312*	*3,826445*
33554432	*4*	*6656*	*1,990355*
33554432	*8*	*3328*	*1,058313*
33554432	*16*	*1664*	*0,589772*
33554432	*32*	*832*	*0,353579*
33554432	*64*	*416*	*0,268133*
33554432	*128*	*208*	*0,226286*
33554432	*256*	*104*	*0,224156*
33554432	*512*	*52*	*0,221182*



Case study n°3 – Texture Memory - Global Memory

In questo caso di studio, è stato analizzato l'algoritmo di Counting Sort utilizzando la memoria globale e la memoria texture; in particolare il vettore da ordinare è stato copiato all'interno della memoria texture, il conteggio delle occorrenze e la riduzione somma vengono effettuate usando la memoria globale.

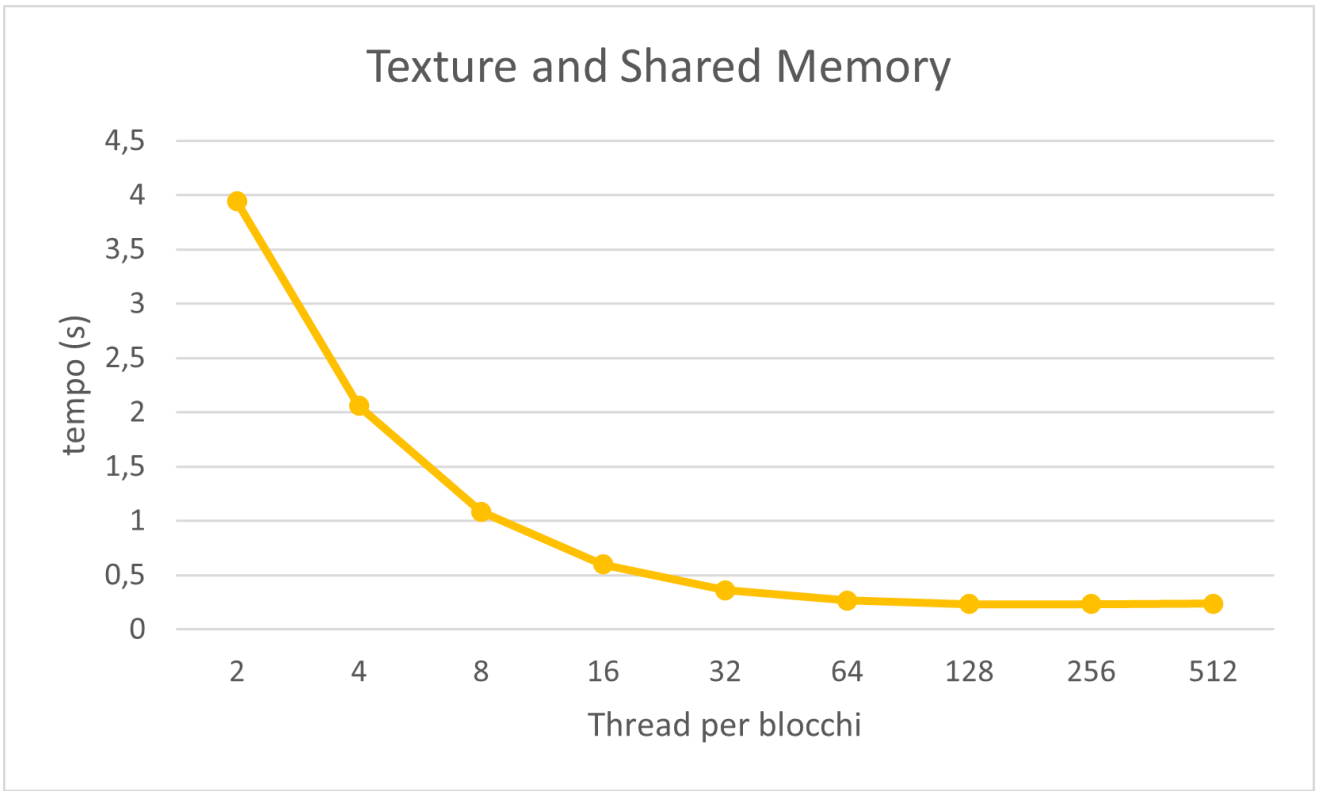
Size	BlockSize	GridSize	Time
33554432	2	13312	5,413774
33554432	4	6656	2,852463
33554432	8	3328	1,539181
33554432	16	1664	0,899531
33554432	32	832	0,575925
33554432	64	416	0,568789
33554432	128	208	0,57304
33554432	256	104	0,554543
33554432	512	52	0,543905



Case study n°4 – Texture Memory – Shared Memory

In questo caso di studio, è stato analizzato l'algoritmo di Counting Sort utilizzando la memoria texture e la memoria shared; in particolare il vettore da ordinare è stato copiato all'interno della memoria texture, il conteggio delle occorrenze e la riduzione somma vengono effettuate usando la memoria shared.

Size	*BlockSize*	*GridSize*	*Time*
33554432	*2*	*13312*	*3,943573*
33554432	*4*	*6656*	*2,058095*
33554432	*8*	*3328*	*1,085679*
33554432	*16*	*1664*	*0,598922*
33554432	*32*	*832*	*0,361101*
33554432	*64*	*416*	*0,266165*
33554432	*128*	*208*	*0,233315*
33554432	*256*	*104*	*0,234269*
33554432	*512*	*52*	*0,236299*



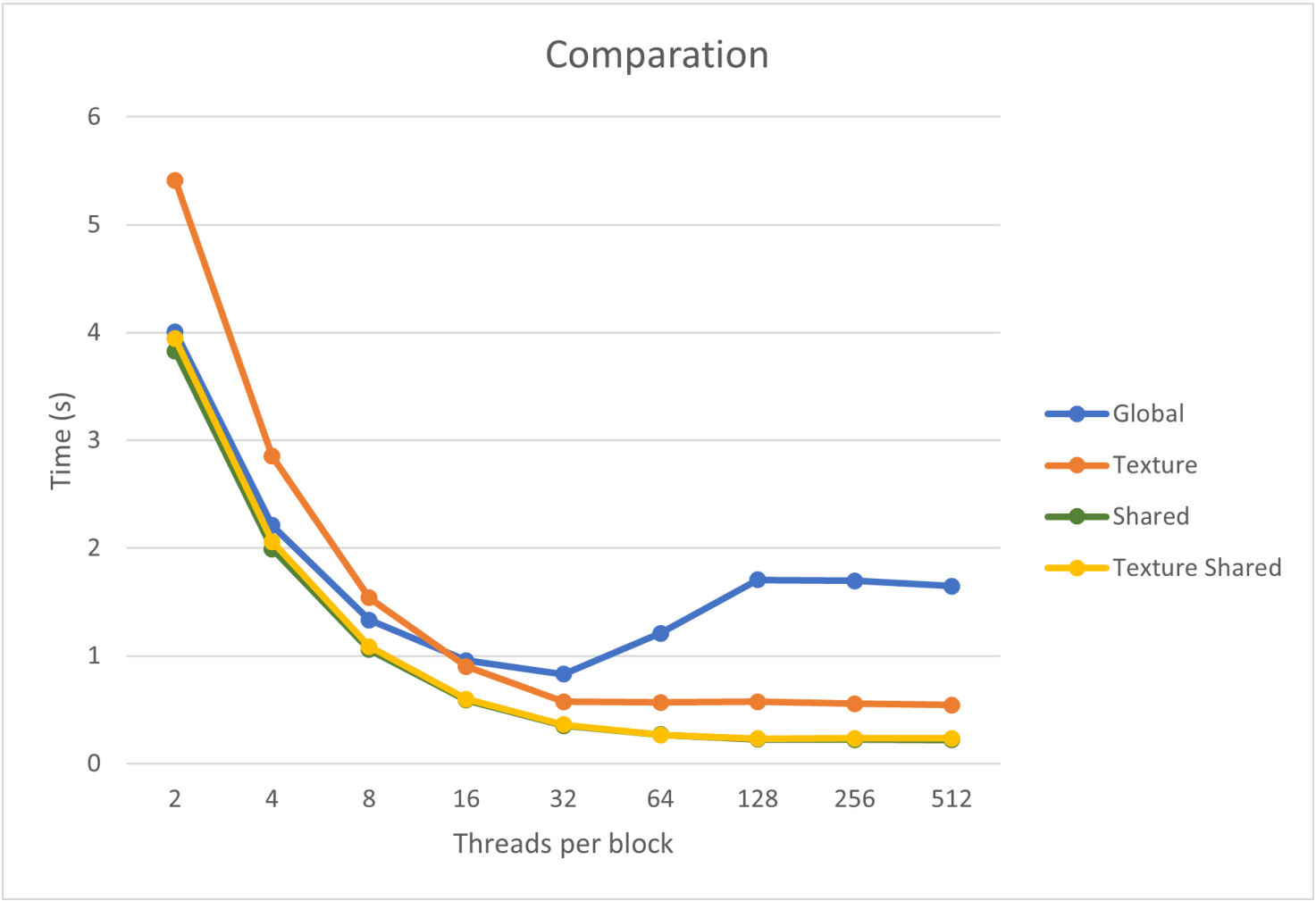
Consideration

L' idea è stata quella di effettuare il conteggio delle occorrenze sulla GPU; in particolare viene eseguito più volte il kernel che si occupa di contare il numero delle occorrenze di un particolare valore all'interno del vettore.
Il kernel è composto da due fasi:

- Una prima fase in cui ogni threads si occupa di controllare se l'elemento corrispondente alla sua cella di memoria è uguale all'elemento passato come paramentro al kernel; infine, il thread incrementa un registro interno se il valore è stato trovato.

- Una seconda fase in cui viene effettuata una riduzione per somma di tutti i contatori dei thread; questa operazione produce in output un vettore di lunghezza pari al numero di blocchi allocati.

Infine, una volta trasferito il vettore alla CPU, essa si occuperà di scrivere in maniera ordinata il valore.



Per il caso di studio N1 (Global) e N3 (Texture-Global), possiamo notare che il tempo di esecuzione è superiore rispetto di4s7p(e)t7t(oi)8a1l(i47no)-7( )5(c)-8(u)-7(i)4( )-nzuaommori

mm-25(o)-7(r)6(i)4ao lobale ha leta te(per motiz nooiz)6, che per re18(u)11((

otei49(o)-7(412(r)6(i)4(s)-8uo)-7li45(t)5(a)-7(601pe)-7(i49ùa)-7(601vs11((e)-7(l)4(o18((c)-8(e)-7.)5(421(m)-2ea)11no)-

How to run

Utilizza il notebook colab nella cartella condivisa di google drive.