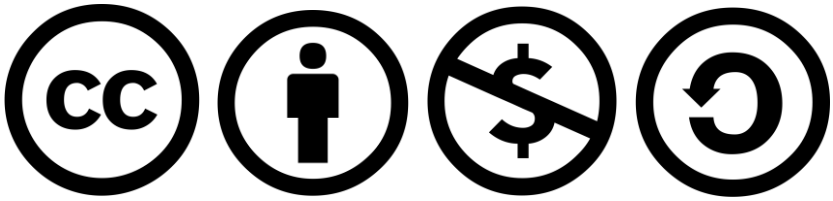


Report Assignment 3

Counting Sort Algorithm

Lecturer: Francesco Moscato - fmoscato@unisa.it

Group:

- Carratù Arianna 0622701696 a.carratu18@studenti.unisa.it
- Di Rienzo Matteo 0622701818 m.dirienzo1@studenti.unisa.it
- Gambardella Giuseppe 0622701666 g.gambardella23@studenti.unisa.it

Index

Problem description

Parallelize and Evaluate Performances of "COUNTING SORT" Algorithm, by using CUDA.

Analyze results with different memory allocation:

1. Global Memory
2. Shared Memory
3. Texture Memory

Counting Sort is an algorithm for sorting a collection of objects according to keys that are small positive integers. It is an integer sorting algorithm. Its worst-case performance is O(n+k), where k is the range of the non-negative key values.

For further information, see: https://en.wikipedia.org/wiki/Counting_sort

Experimental setup 1

Hardware

CPU

```
processor: 0
vendor_id: GenuineIntel
cpu family: 6
model: 63
model name: Intel(R) Xeon(R) CPU @ 2.30GHz
stepping: 0
microcode: 0x1
cpu MHz: 2299.998
cache size: 46080 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
invpcid_single ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid xsaveopt arat md_clear arch_capabilitiesbugs:
cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
bogomips: 4599.99
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: 63
model name: Intel(R) Xeon(R) CPU @ 2.30GHz
stepping: 0
microcode: 0x1
cpu MHz: 2299.998
cache size: 46080 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
invpcid_single ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid xsaveopt arat md_clear arch_capabilitiesbugs:
cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs
bogomips: 4599.99
clflush size: 64
cache_alignment: 64
address sizes: 46 bits physical, 48 bits virtual
power management:
```

GPU

```
Device number: 0
      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
```

- The Tesla K80 has a maximum of 16 blocks for SM and one SM can have a maximum 1024 threads.

Bandwidth

```
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: Tesla T4
 Range Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)    Bandwidth(GB/s)
   1000                     0.4
   101000             9.3
   201000             8.3
   301000             8.3
   401000             8.8
   501000             8.8
   601000             8.9
   701000             8.9
   801000             9.1
   901000             9.0

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)    Bandwidth(GB/s)
   1000                     0.5
   101000                   10.9
   201000                   9.6
   301000                   9.8
   401000                   9.6
   501000                   9.2
   601000                   9.1
   701000                   9.2
   801000                   9.5
   901000                   9.6

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)    Bandwidth(GB/s)
   1000                     0.6
   101000                   53.3
   201000                   87.4
```

```
301000            123.5
401000            144.8
501000            162.7
601000            182.8
701000            192.3
801000            209.2
901000            219.2
```

Software

- Google Colab

Experimental setup 2

CPU

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz
stepping        : 3
microcode       : 0xffffffff
cpu MHz         : 3400.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt
```

```
tsc_deadline_timer aes xsave osxsave avx f16c rdrand lahf_lm abm
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid ibrs ibpb stibp
ssbd
bogomips        : 6800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz
stepping        : 3
microcode       : 0xffffffff
cpu MHz         : 3400.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt
tsc_deadline_timer aes xsave osxsave avx f16c rdrand lahf_lm abm
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid ibrs ibpb stibp
ssbd
bogomips        : 6800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 2
```

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz
stepping        : 3
microcode       : 0xffffffff
cpu MHz         : 3400.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt
tsc_deadline_timer aes xsave osxsave avx f16c rdrand lahf_lm abm
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid ibrs ibpb stibp
ssbd
bogomips        : 6800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz
stepping        : 3
microcode       : 0xffffffff
cpu MHz         : 3400.000
cache size      : 256 KB
physical id     : 0
siblings        : 4
```

```
core id         : 1
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt
tsc_deadline_timer aes xsave osxsave avx f16c rdrand lahf_lm abm
fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid ibrs ibpb stibp
ssbd
bogomips        : 6800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

GPU

```
 Device name: NVIDIA GeForce GTX 750 Ti
  Compute capability: 5.0

  Clock Rate: 1137000 kHz
  Total SMs: 5
  Shared Memory Per SM: 65536 bytes
  Registers Per SM: 65536 32-bit
  Max threads per SM: 2048
  L2 Cache Size: 2097152 bytes
  Total Global Memory: 2147221504 bytes
  Memory Clock Rate: 2700000 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
```

Bandwidth

```
Device 0: NVIDIA GeForce GTX 750 Ti
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     6547.7

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     6541.3

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     74511.7

Result = PASS
```

Software

- CUDA 9.2
- MSVC for Windows

Performance

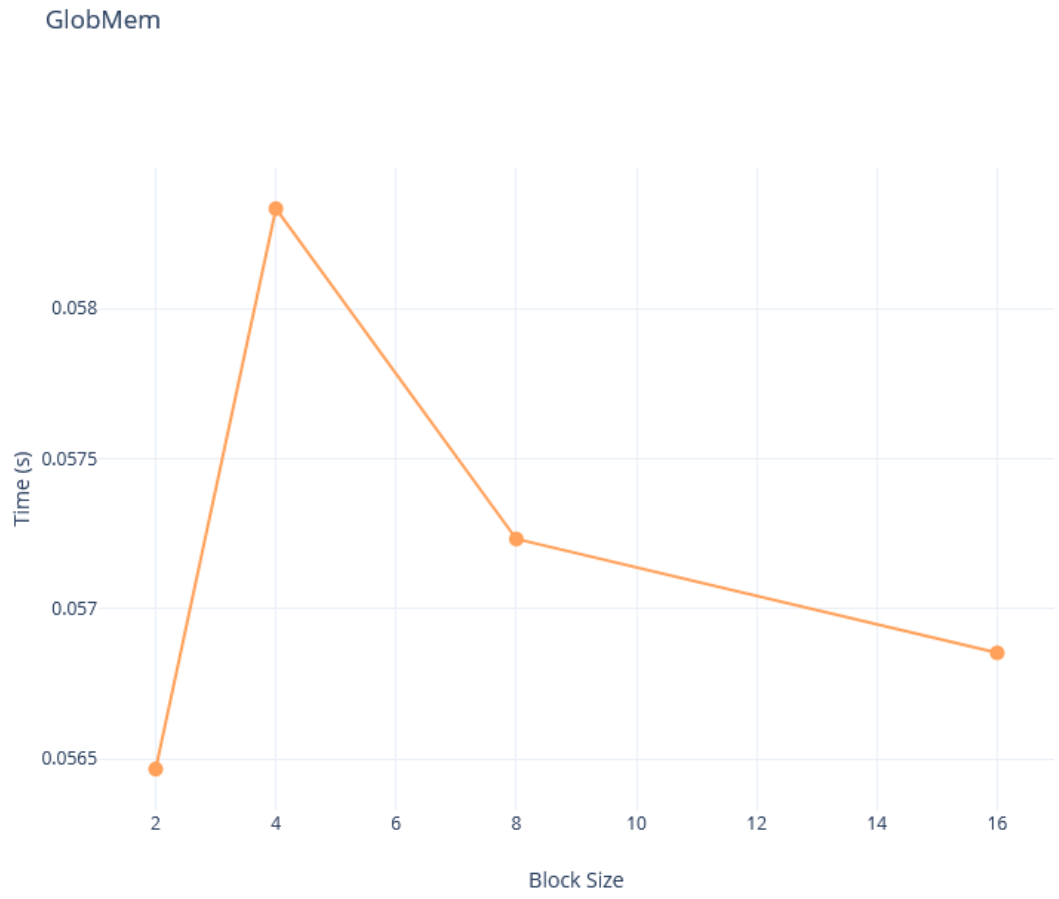
Case n° 1 - Global Memory

In this case study, we have analyzed the problem of counting sort algorithm using global memory. The size of the problem is 2^25 (33.554.432) and then we evaluate the performances with block size of:

- 128: with this choice we have that 2048/128 = 16 blocks. The occupancy is 100%.
- 256: with this choice we have that 2048/256 = 8 blocks. The occupancy is 100%.
- 512: with this choice we have that 2048/512 = 4 blocks. The occupancy is 100%.
- 1024: with this choice we have that 2048/1024= 2 blocks. The occupancy is 100%.

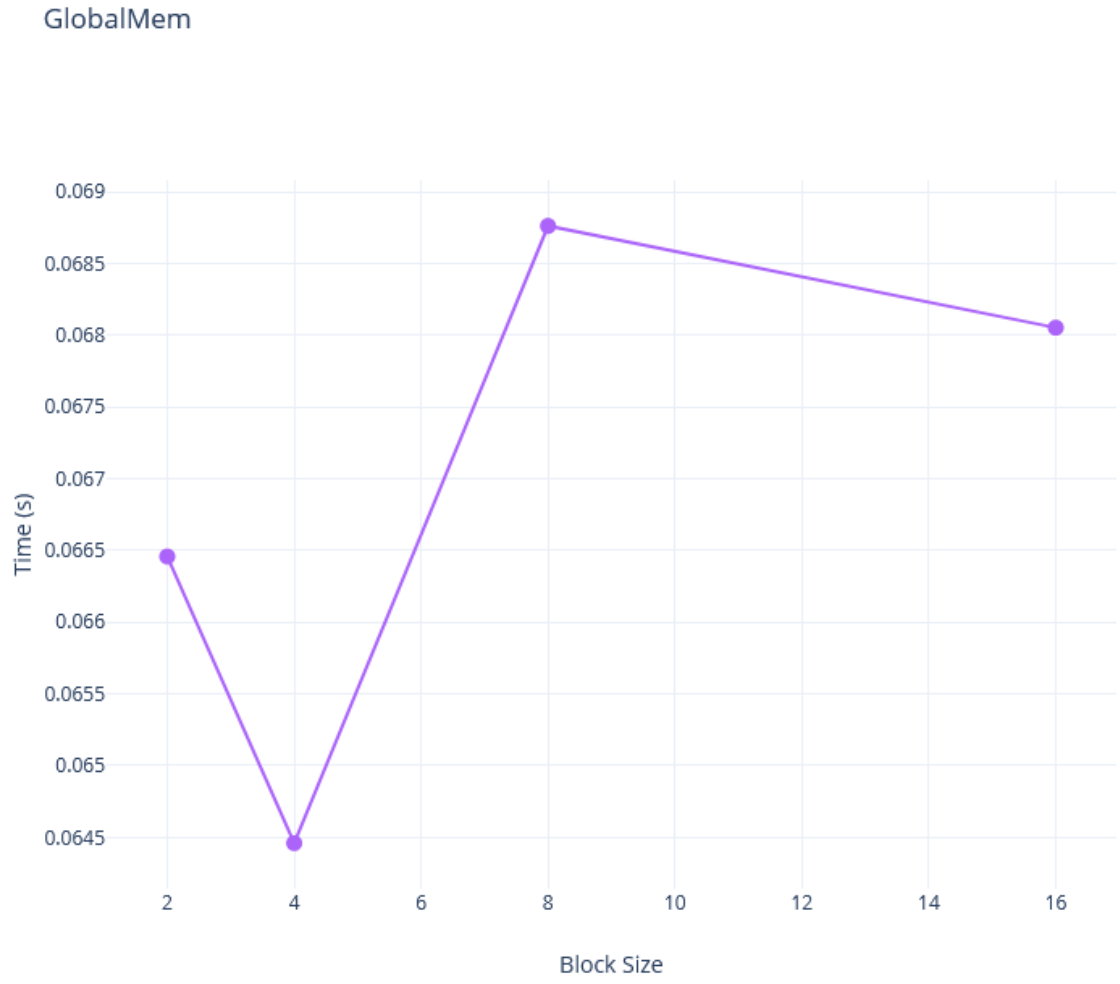
Experimental Setup 1

N	TotalTime	BlockSize	GridSize	Gflops
33.554.432	0.055289s	128	262144	0.0003
33.554.432	0.057232s	256	131072	0.0005
33.554.432	0.058332s	512	65536	0.0010
33.554.432	0.056466s	1024	32768	0.0020



Experimental Setup 2

N	TotalTime	BlockSize	GridSize	Gflops
33.554.432	0.068053s	128	262144	0.0001
33.554.432	0.068762s	256	131072	0.0003
33.554.432	0.064460s	512	65536	0.0006
33.554.432	0.066458s	1024	32768	0.0012



Case n° 2 - Shared Memory

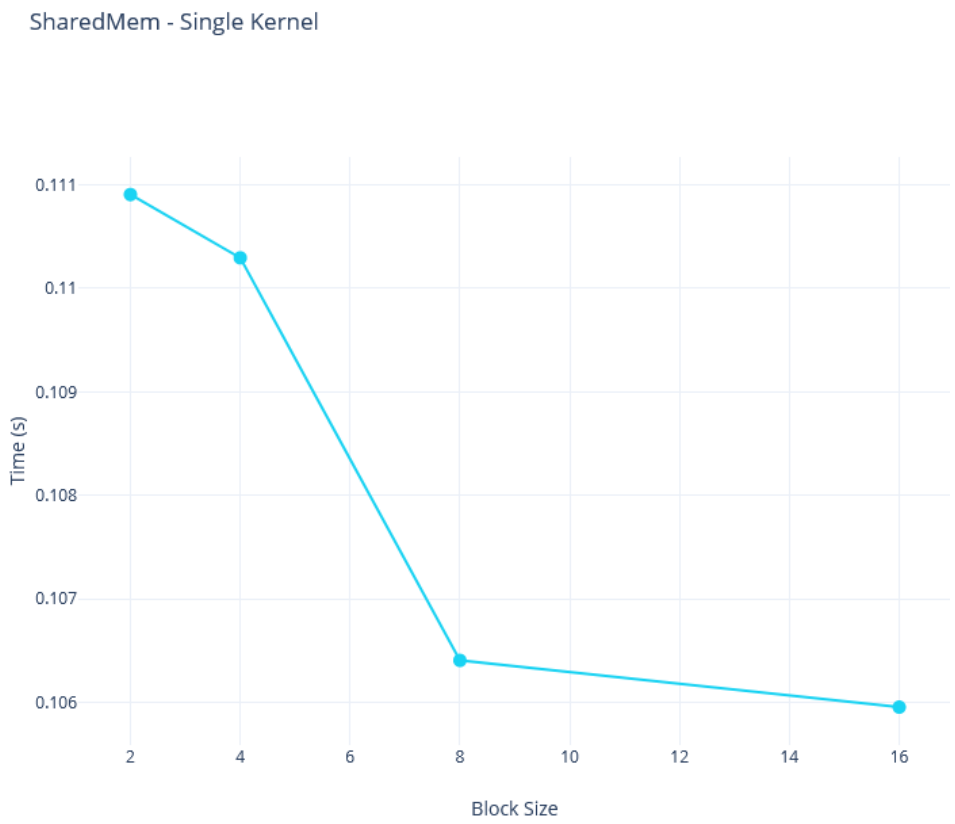
In this case study, we have analyzed the problem of counting sort algorithm using shared memory. We implemented two different ways to calculate the minimum and the maximum value in the array: the first one computes minimum and maximum in the same kernel, the second one utilizes two different kernels that are ran simultaneously.

- 128: with this choice we have that 2048/128 = 16 blocks. The occupancy is 100%.
- 256: with this choice we have that 2048/256 = 8 blocks. The occupancy is 100%.
- 512: with this choice we have that 2048/512 = 4 blocks. The occupancy is 100%.
- 1024: with this choice we have that 2048/1024= 2 blocks. The occupancy is 100%.

First Case: Single kernel for minimum and maximum

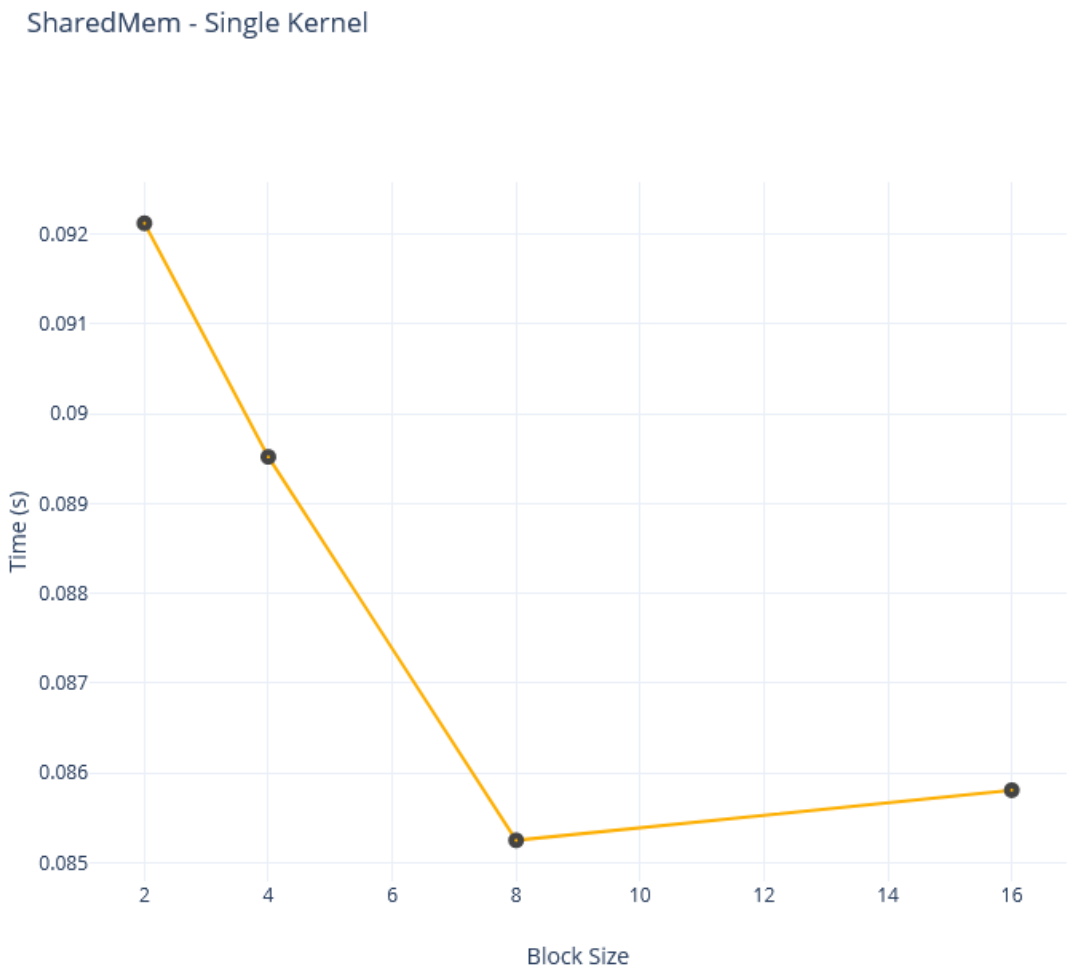
Experimental Setup 1

N	TotalTime	BlockSize	GridSize	Gflops MaxMin	Gflops Counting
33.554.432	0.105956s	128	262144	0.0003	0.0024
33.554.432	0.106407s	256	131072	0.0039	0.0005
33.554.432	0.110295s	512	65536	0.0061	0.0011
33.554.432	0.110906s	1024	32768	0.0090	0.0021



Experimental Setup 2

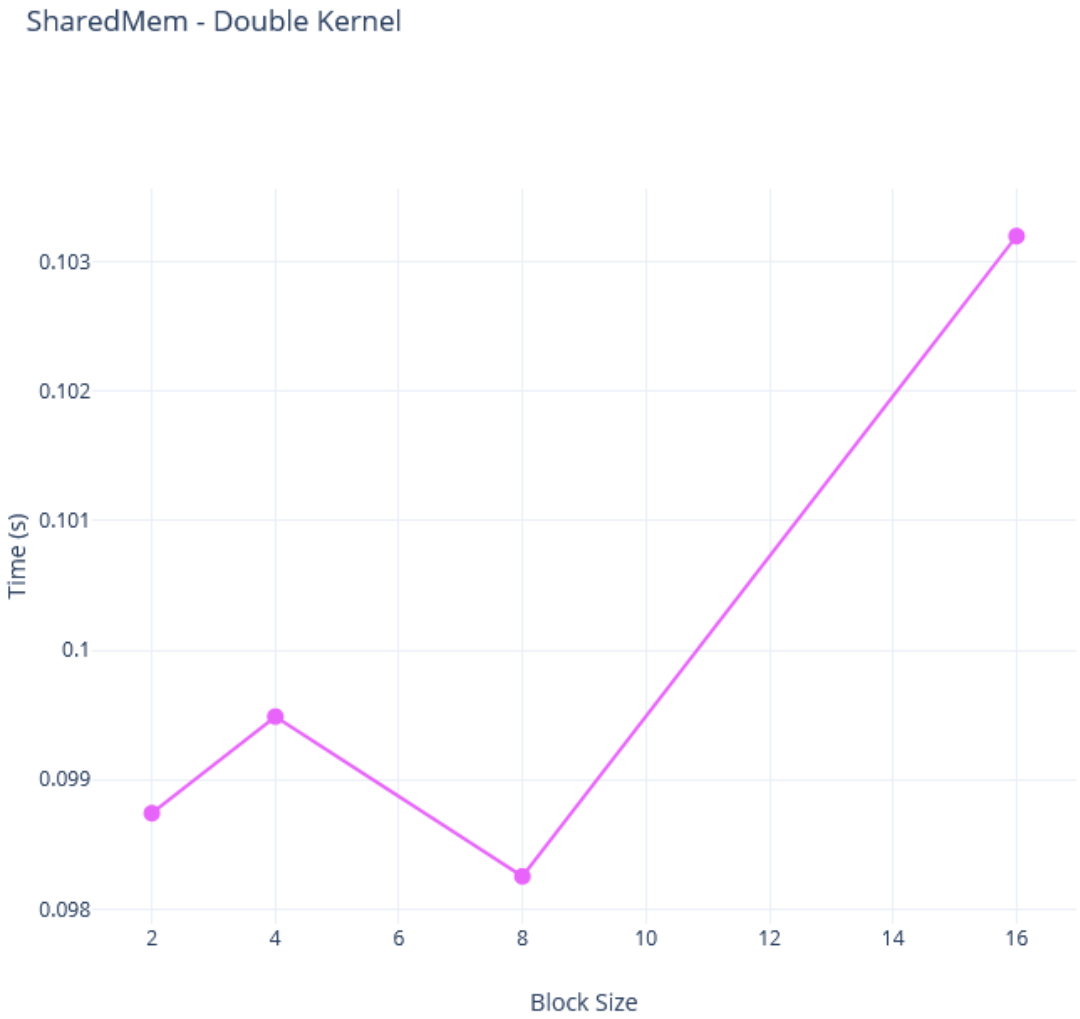
N	TotalTime	BlockSize	GridSize	Gflops MaxMin	Gflops Counting
33.554.432	0.085810s	128	262144	0.0014	0.0001
33.554.432	0.085252s	256	131072	0.0026	0.0003
33.554.432	0.089520s	512	65536	0.0043	0.0005
33.554.432	0.092121s	1024	32768	0.0074	0.0012



Second Case: Double kernel for minimum and maximum

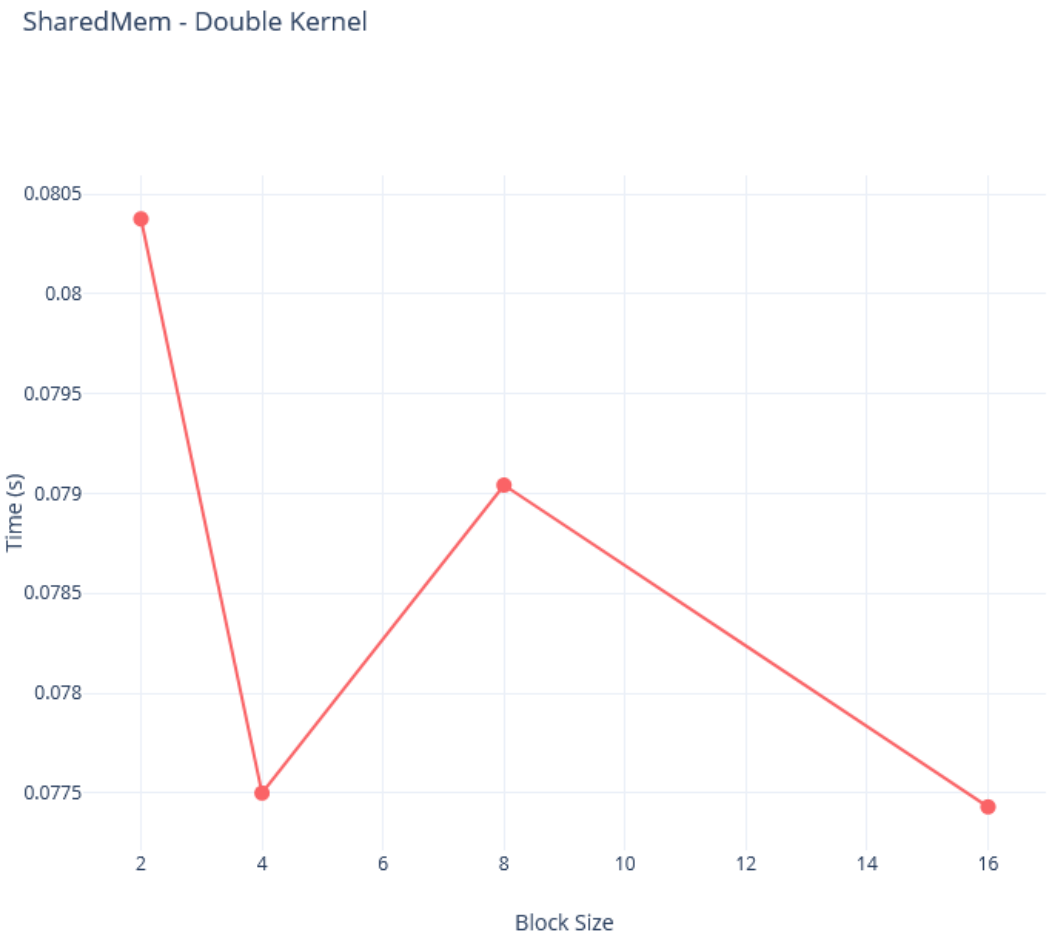
Experimental Setup 1

N	TotalTime	BlockSize	GridSize	Gflops Max	Gflops Min	Gflops Counting
33.554.432	0.103198s	128	262144	0.0002	0.0002	0. 0003
33.554.432	0.098253s	256	131072	0.0008	0.0008	0.0006
33.554.432	0.099487s	512	65536	0.0014	0.0014	0.0011
33.554.432	0.098741s	1024	32768	0.0023	0.0023	0.0021



Experimental Setup 2

N	TotalTime	BlockSize	GridSize	Gflops Max	Gflops Min	Gflops Counting
33.554.432	0.077431s	128	262144	0.0002	0.0002	0.0001
33.554.432	0.079042s	256	131072	0.0005	0.0005	0.0003
33.554.432	0.077500s	512	65536	0.0009	0.0009	0.0006
33.554.432	0.080375s	1024	32768	0.0014	0.0014	0.0012

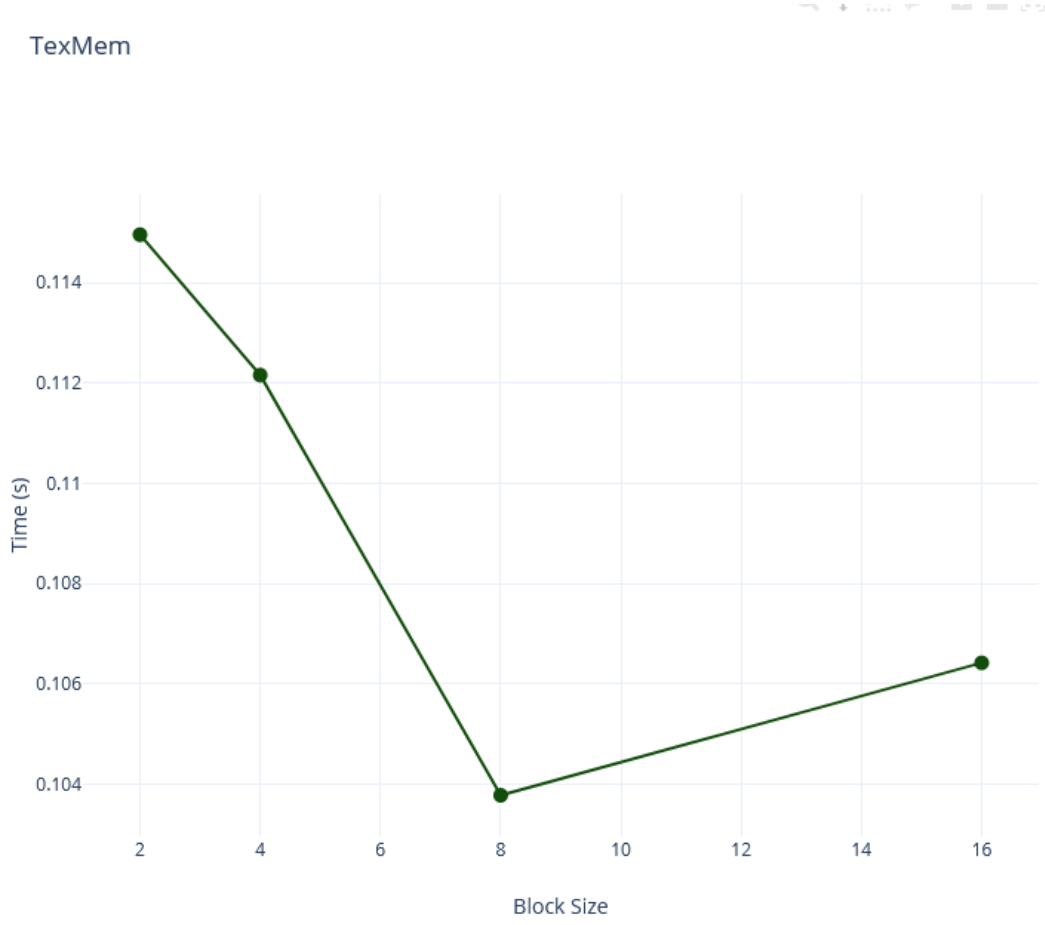


Case n° 3 - Texture Memory

- In this case study, we have analyzed the problem of counting sort algorithm using texture memory. The size of the problem is 2^25 (33.554.432) and then we evaluate the performances with block size of:
-
- 128 : with this choice we have that 2048/128 = 16 blocks. The occupancy is 100%.
- 256: with this choice we have that 2048/256 = 8 blocks. The occupancy is 100%.
- 512: with this choice we have that 2048/512 = 4 blocks. The occupancy is 100%.
- 1024: with this choice we have that 2048/1024= 2 blocks. The occupancy is 100%.

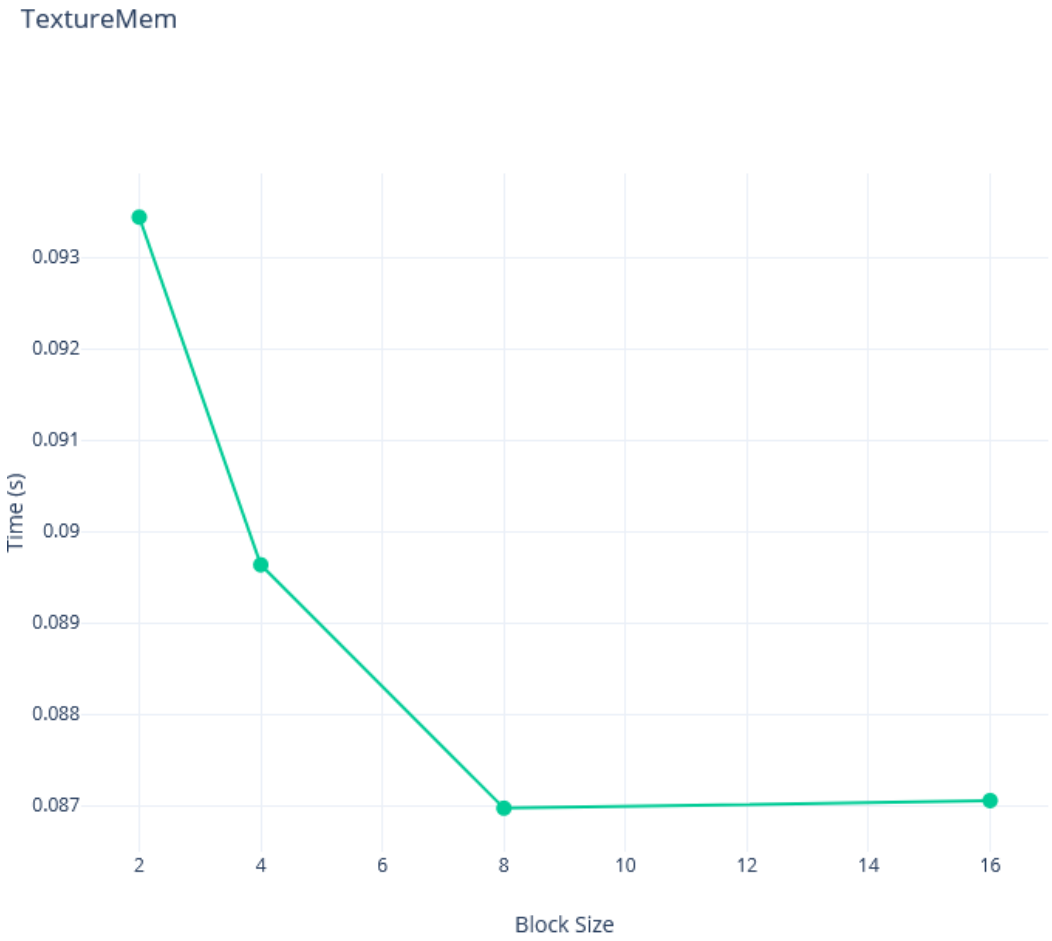
Experimental Setup 1

N	TotalTime	BlockSize	GridSize	Gflops Max and Min	Gflops Counting
33.554.432	0.106417s	128	262144	0.0024	0.0003
33.554.432	0.103774s	256	131072	0.0038	0.0005
33.554.432	0.112158s	512	65536	0.0059	0.0010
33.554.432	0.114959	1024	32768	0.0087	0.0020



Experimental Setup 2

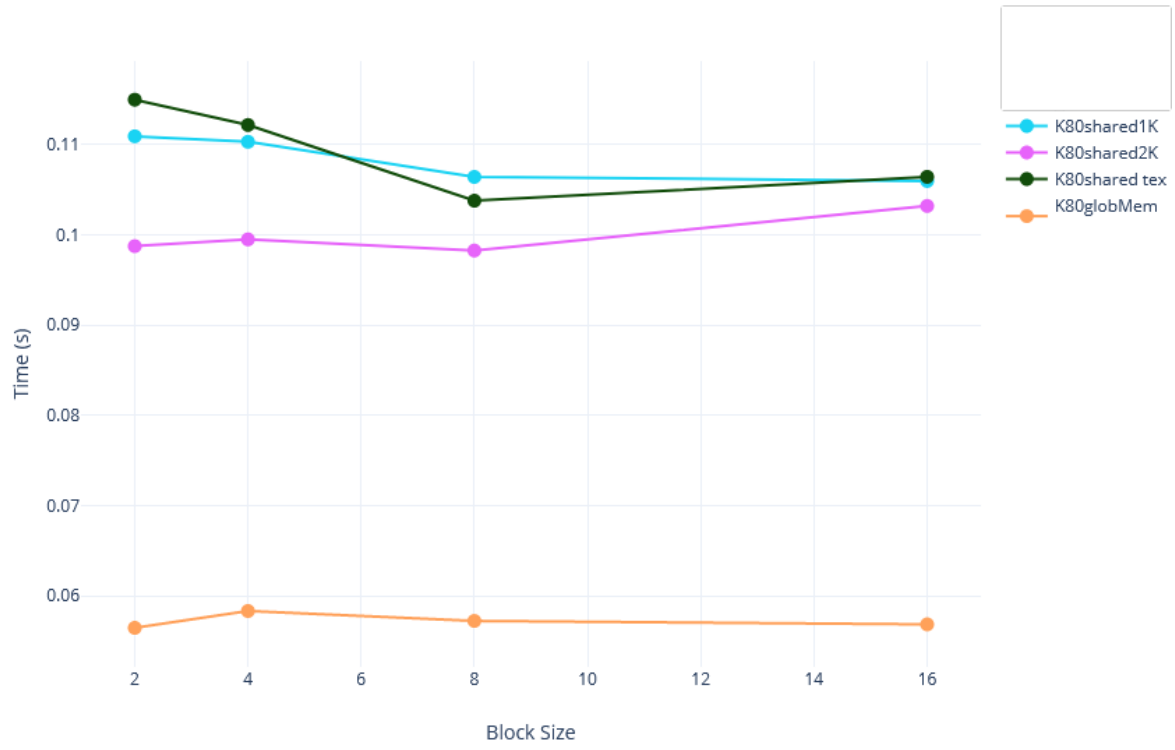
N	TotalTime	BlockSize	GridSize	Gflops Max and Min	Gflops Counting
33.554.432	0.087055s	128	262144	0.0015	0.0001
33.554.432	0.086972s	256	131072	0.0027	0.0003
33.554.432	0.089636s	512	65536	0.0045	0.0006
33.554.432	0.093441s	1024	32768	0.0076	0.0012



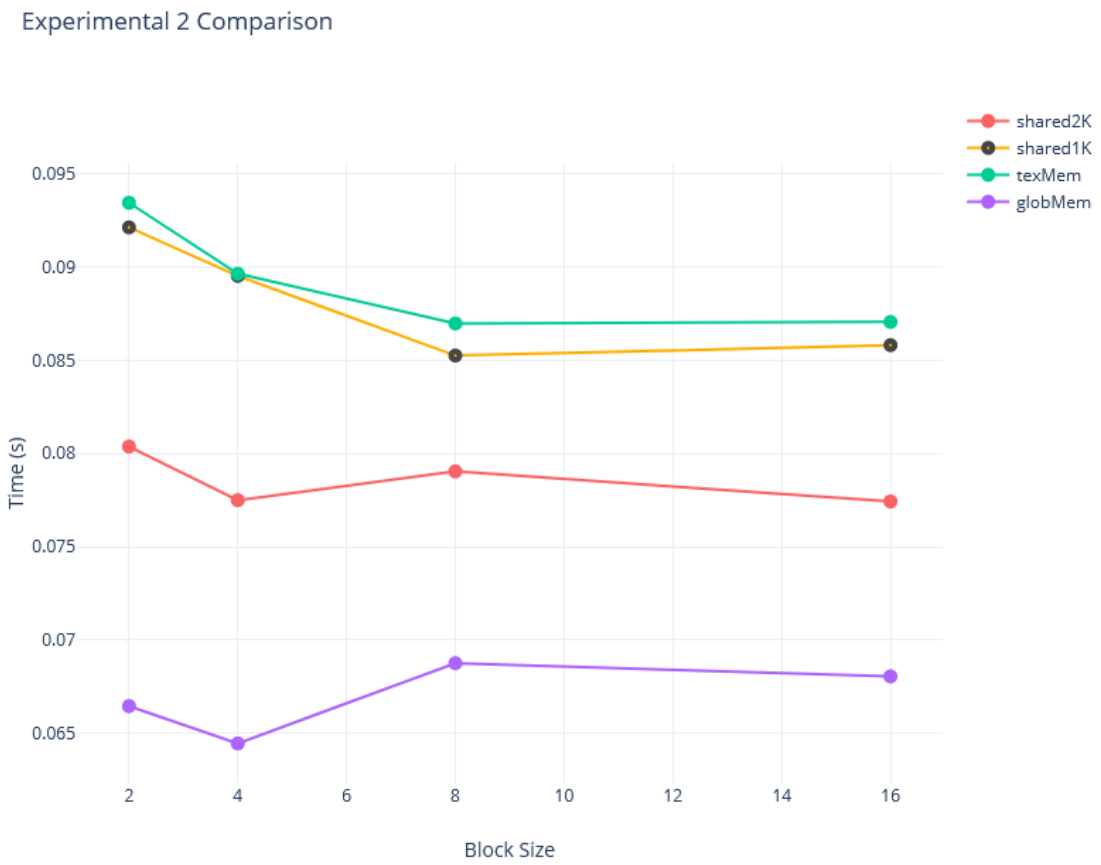
Considerations

Experimental Setup 1

Experimental 1 Comparison



Experimental Setup 2



Premises

- GPU architecture (pre-Fermi cards have no caches other than texture cache so that using texture memory should give better memory performance)
- Global memory is L1/L2-cache and texture memory is a separate texture cache
- Memory access pattern: We read integer, each cache has a different cache line width and behaves differently for different data types
- L2 is global so each access to global memory goes through L2 first, L1 is local to multiprocessor
- Texture memory is read-only, our program cannot dictate what is in the texture cache, so it is transparent to the program. (behaves much like a CPU cache)

- Counting sort does not benefit from locality: each thread needs to be aware if there is another thread writing in the same memory's area in the second phase of algorithm (calculating counting array). Max and min problem is not a local problem too, because we need to do a synchronization among all threads results.

The best performances that we can achieve with this type of problem are with the global memory, the reason is that in this case we calculate max and min sequentially in CPU. So there is no need to synchronize threads with __*synchthreads()* primitive, moreover both max and min kernels if run with many elements cause a long wait for synchronization.

Global memory has high latency due to technological reasons, also due to the presence of the L2 cache (since every access pass through the cache we can also have cache misses). It is the best choice for this algorithm, since we can easily synchronize memory conflicts with *atomicAdd* primitive. We also experimented that the kernelCounting array is softly faster with a support shared memory.

Texture memory is a read-only cache, therefore we can have cache misses but in general it has little better performance than the global memory. In this implementation it seems slower because of maxmin reduction.

Furthermore, comparing the two different experimental setups we note that:

- the first experimental setup (K80) is a little bit faster than the second one (GTX 750 Ti) with the global memory
- the second experimental setup is faster (about 20%) than the first one with the two kernels shared memory and with the one kernel shared memory
- the second experimental setup is faster (about 20%) than the first one with the texture memory

  We suppose the subtle differences between the two experimental setups are due to some different gpu's characteristics.

Test case

Testing was done by sorting the same array on the host device and checking element by element the correctness of the result.

How to run

1. Create a folder “cuda_src” in Google Drive
2. Copy the .cu files into “cuda_src”
3. Open the .ipynb notebook in Google Colab
4. Go to Runtime>Run after