Report Common Assignment 3

Counting sort

Rosa Gerardo

Scovotto Luigi

Tortora Francesco

January, 2022

Index

Chapter 1

Problem description

Parallelize and Evaluate Performances of "Counting Sort" Algorithm , by using CUDA.

Counting sort is an algorithm for sorting integer numbers. The computational complexity is equal to $O(n)$.

In this report we analize counting sort algorithm: get execution times, bandwidth and other performances measures when of using:

- Global Memory
- Shared Memory
- Texture Memory

Evaluate performances with different block-grid configurations.

1.1 Experimental setup

All measurements were made using the Google Colab service, with the following setup.

1.1.1 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_capabilities
bugs                    : 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_capabilities
bugs                : 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:
```

RAM

```
MemTotal:        13302920 kB
MemFree:           236348 kB
MemAvailable:    12350076 kB
Buffers:           283016 kB
Cached:          11566856 kB
SwapCached:             0 kB
Active:           3457004 kB
Inactive:         8911716 kB
Active(anon):      476972 kB
Inactive(anon):       480 kB
Active(file):     2980032 kB
Inactive(file):   8911236 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:              0 kB
SwapFree:               0 kB
Dirty:                200 kB
Writeback:              0 kB
AnonPages:         518888 kB
Mapped:            269548 kB
Shmem:               1228 kB
KReclaimable:      548408 kB
Slab:              600016 kB
SReclaimable:      548408 kB
SUnreclaim:         51608 kB
KernelStack:         5904 kB
PageTables:          7172 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
```

```
CommitLimit:      6651460 kB
Committed_AS:     3798912 kB
VmallocTotal:   34359738367 kB
VmallocUsed:        46004 kB
VmallocChunk:           0 kB
Percpu:              1448 kB
AnonHugePages:          0 kB
ShmemHugePages:         0 kB
ShmemPmdMapped:         0 kB
FileHugePages:          0 kB
FilePmdMapped:          0 kB
CmaTotal:               0 kB
CmaFree:                0 kB
HugePages_Total:        0
HugePages_Free:         0
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
Hugetlb:                0 kB
DirectMap4k:       279360 kB
DirectMap2M:      9154560 kB
DirectMap1G:      6291456 kB
```

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
```

BANDWIDTH

```
Device 0: Tesla K80
Range Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  1000                         278.7
  101000                       6240.3
  201000                       6613.4
  301000                       6923.8
  401000                       7064.9
  501000                       7197.9
  601000                       7046.2
  701000                       7217.7
  801000                       7333.4
  901000                       7454.2

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  1000                         419.3
  101000                       6563.5
  201000                       7137.2
  301000                       7351.7
  401000                       7361.0
  501000                       7443.2
  601000                       7505.5
  701000                       7519.7
  801000                       7579.2
  901000                       7571.1
```

```
Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  1000                         251.6
  101000                       25527.1
  201000                       44132.9
  301000                       57341.3
  401000                       75400.8
  501000                       83089.5
  601000                       89331.3
  701000                       102785.9
  801000                       91342.3
  901000                       80354.0
```

1.1.2 Software

- NVIDIA-SMI 510.39.01
- Driver Version: 460.32.03
- CUDA Version: 11.2
- Ubuntu 18.04.5 LTS

Chapter 2

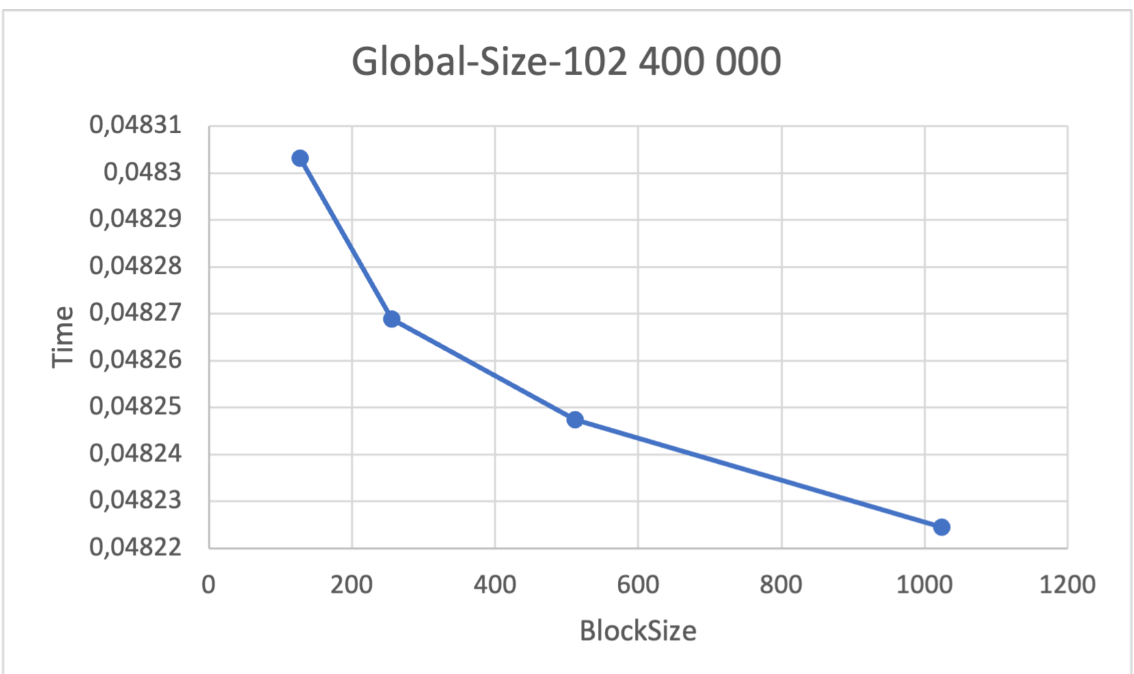
Performance

2.1 Case study n°1 - Global Memory

In this case study, we have analyzed the counting sort algorithm using global memory. The size of the problem is 102 400 000 and than 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%.

N	BlockSize	GridSize	gips	time_sec
102 400 000	128	800 000	40 278 885	0,04830324
102 400 000	256	400 000	40 307 517	0,04826894
102 400 000	512	200 000	40 325 467	0,04824748
102 400 000	1024	100 000	40 344 692	0,04822454

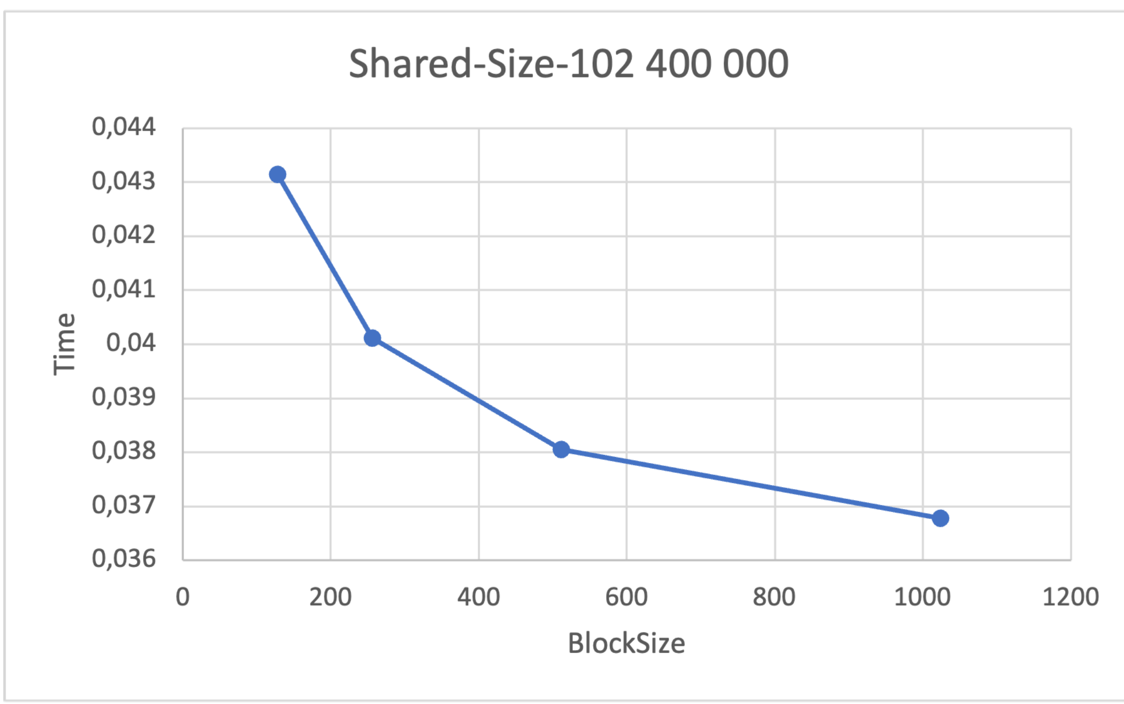


2.2 Case study n°2 - Shared Memory

In this case study, we have analyzed the counting sort algorithm using shared memory. The size of the problem is 102 400 000 and than 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%.

N	BlockSize	GridSize	gips	time_sec
102 400 000	128	800 000	52 675 126	0,04314758
102 400 000	256	400 000	57 151 101	0,04011825
102 400 000	512	200 000	61 304 339	0,03805279
102 400 000	1024	100 000	65 612 982	0,03677322

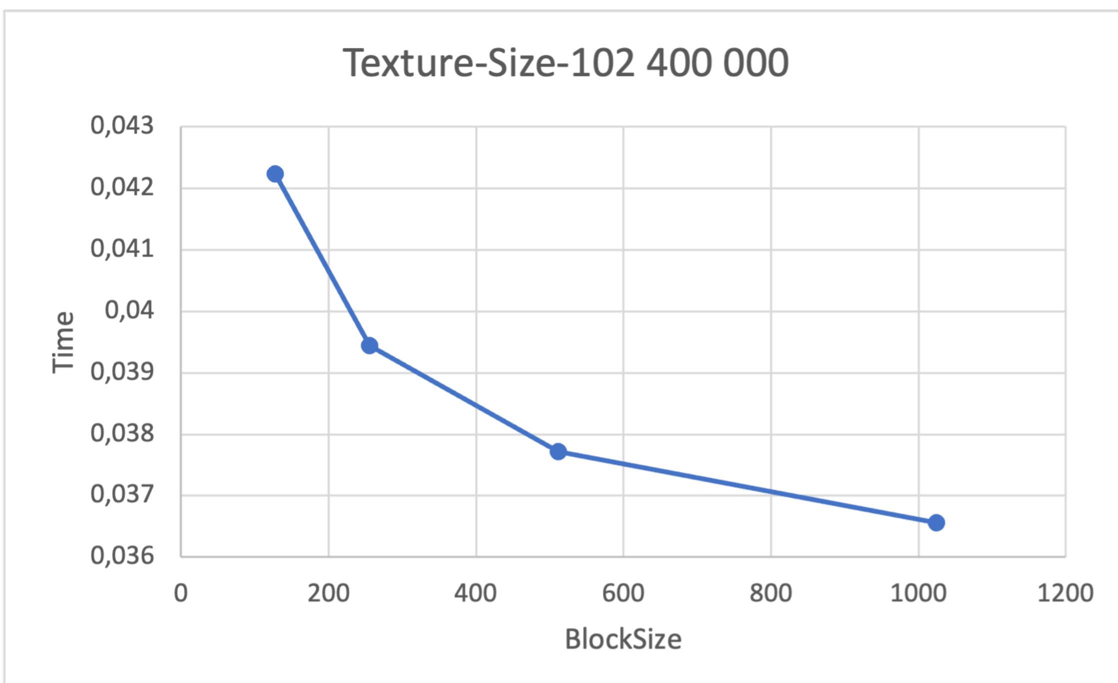


2.3 Case study n°3 - Texture Memory

In this case study, we have analyzed the counting sort algorithm using texture memory. The size of the problem is 102 400 000 and than 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%.

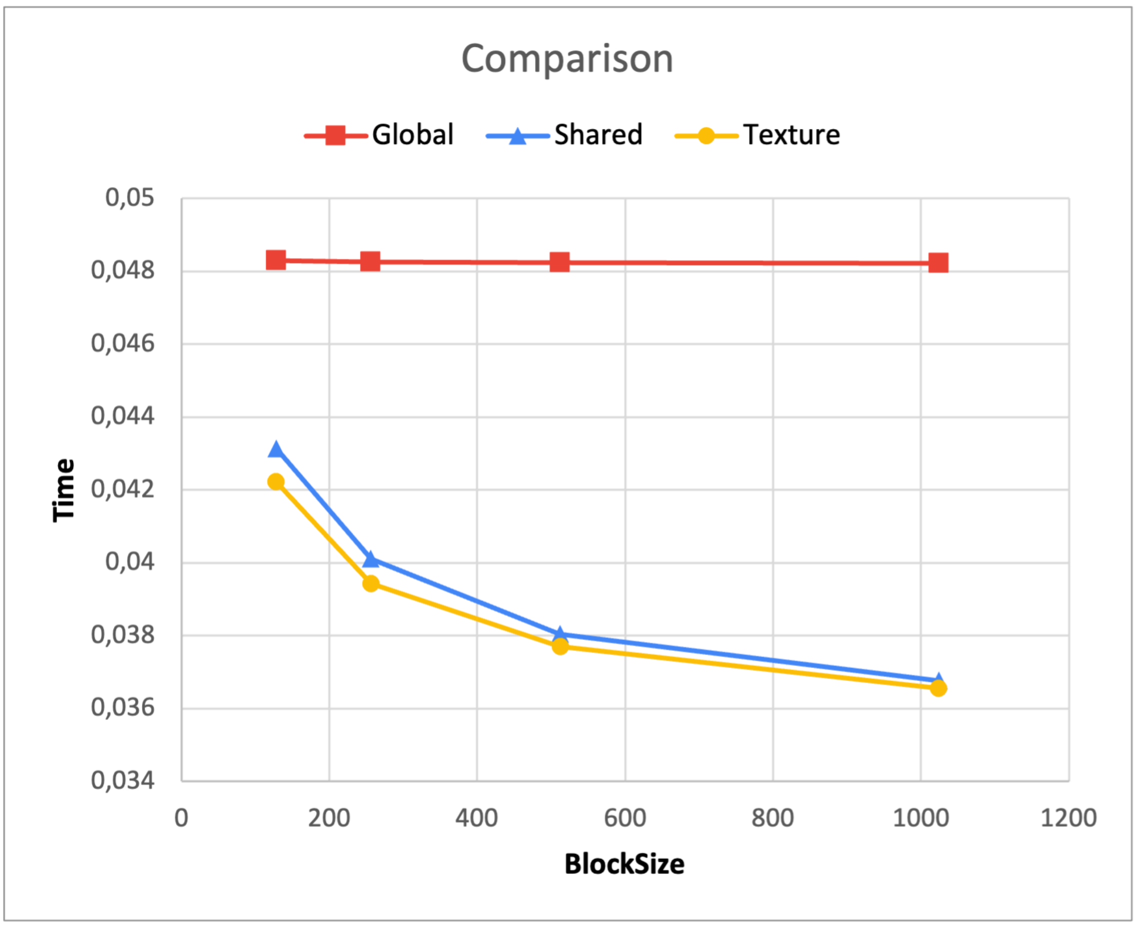
N	BlockSize	GridSize	gips	time_sec
102 400 000	128	800 000	53 808 727	0,04223858
102 400 000	256	400 000	58 131 895	0,03944138
102 400 000	512	200 000	61 855 504	0,03771372
102 400 000	1024	100 000	65 988 295	0,03656407



Chapter 3

Considerations

3.1 Comparison



As can be seen from the graph above, shared and textured versions are influenced by the BlockSize: as the size increases, the time decreases.

Comparing the various versions, a clear improvement can be seen in the use of shared compared to global as the shared has a very low latency. In the texture, there is an improvement over the global and shared versions, but only a small improvement in terms of time compared to the shared.

3.2 Premises

- Global memory is L1/L2-cache and texture memory is a separate texture cache.
- L2 is global so each access to global memory goes through L2 first, L1 is local to the 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).
- We used atomic functions because the counting sort algorithm required that several threads access the same locations but above all it avoids data inconsistency due to the race condition.

  Referring to the documentation, we realized that the atomic function was useful for our purpose because:

  an atomic function performs a read-modify-write atomic operation on one 32-bit or 64-bit word residing in global or shared memory.

  The operation is atomic in the sense that it is guaranteed to be performed without interference from other threads. In other words, no other thread can access this address until the operation is complete.

  Atomic functions do not act as memory fences and do not imply synchronization or ordering constraints for memory operations.

  In particular, we used atomicAdd and atomicSub function, the following URLs are referred to the specified functions documentation:

  https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomicadd

  https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomicsub

Chapter 4

How to run

Use the colab notebook in the folder or go to this URL:

https://drive.google.com/drive/u/1/folders/10ZMNoKnH-foAqqGiVjpBPBVHRx-Z5TAw

You can find the complete project on GitHub: https://github.com/scov8/CommonAssignment3

The starting point for the global and shared counting sort was taken here:

https://github.com/avtrunov/CUDA/blob/500bf0f1ca9af066f8dba6572d7b11121046d3a6/CountingSort.cu

https://github.com/jtruong27/ECEC413/blob/9713c61a9c5dc10c77b5d79def7ea83abc21fcb0/Homework_5/CountingSort_CUDA/counting_sort_kernel.cu

The previous year's group 02 files proposed by the professor during the course were used for file generation and extraction.