

# Assignment I:

## GPU architecture and basic programming environments

Christian Weigelt, Gábor Nagy

November 9, 2023

### Contributions

We cooperated and worked together on the exercises. When it came to doing exercise 3, we encountered difficulties running and compiling the benchmarks through WSL2 on Christian's computer, so we opted to use a native Ubuntu installation instead, using Gábor's computer, which is fitted with an NVIDIA GTX 1070 GPU.

### Exercises

#### Exercise 1

*List the main differences between GPUs and CPUs in terms of architecture.*

- CPU is latency-oriented, while GPUs are throughput-oriented, i.e. focus is on running many tasks in parallel, instead of running them quickly.
  - To achieve this, CPUs have a small number of very fast cores (high clock rate), while GPUs feature a large number of simpler, slower processing cores.
- CPUs can run instructions out of order to decrease latency, while GPUs always execute instructions in the order they appear.
- CPU tasks can be related, and might need synchronisation for correct results, while GPUs execute independent tasks in parallel, eliminating the need for synchronisation.
  - This inherently also limits the possibility of synchronisation on GPUs, which means more care has to be taken when partitioning work, in order to logically isolate computations.

*Check the latest Top500 list that ranks the top 500 most powerful supercomputers in the world. In the top 10, how many supercomputers use GPUs? Report the name of the supercomputers and their GPU vendor (Nvidia, AMD, ...) and model.*

1. Frontier - AMD INSTINCT MI250X GPUs
2. LUMI - AMD INSTINCT MI250X GPUs
3. Leonardo - NVIDIA A100 XSM4 64GB GPUs
4. Summit - NVIDIA Volta GV100 GPUs
5. Sierra - NVIDIA Volta GV100 GPUs
6. Perlmutter - NVIDIA A100 XSM4 40GB GPUs
7. Selene - NVIDIA A100 GPUs

*One main advantage of GPU is its power efficiency, which can be quantified by Performance/Power, e.g., throughput as in FLOPS per watt power consumption. Calculate the power efficiency for the top 10 supercomputers. (Hint: use the table in the first lecture)*

1. Frontier: 52.59 GFlops/Watts
2. Supercomputer Fugaku: 14.78 GFlops/Watts
3. LUMI: 51.38 GFlops/Watts
4. Leonardo: 32.24 GFlops/Watts
5. Summit: 14.72 GFlops/Watts
6. Sierra: 12.72 GFlops/Watts
7. Sunway TaihuLight: 6.05 GFlops/Watts
8. Perlmutter: 27.37 GFlops/Watts
9. Selene: 23.98 GFlops/Watts
10. Tianhe-2A: 3.32 GFlops/Watts

## Exercise 2

Output of *deviceQuery* in */1-Utilities*:

```
CUDA Device Query (Runtime API) version (CUDA static linking)
Detected 1 CUDA Capable device(s)
Device 0: "NVIDIA GeForce GTX 1060"
  CUDA Driver Version / Runtime Version      12.3 / 12.3
  CUDA Capability Major/Minor version number: 6.1
  Total amount of global memory:              6144 MBytes (6442254336 bytes)
  (010) Multiprocessors, (128) CUDA Cores/MP: 1280 CUDA Cores
  GPU Max Clock rate:                        1733 Mhz (1.73 GHz)
  Memory Clock rate:                          4804 Mhz
  Memory Bus Width:                           192-bit
  L2 Cache Size:                             1572864 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:             65536 bytes
  Total amount of shared memory per block:     49152 bytes
  Total shared memory per multiprocessor:      98304 bytes
  Total number of registers available per block: 65536
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:         1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                      2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:        Yes with 1 copy engine(s)
  Run time limit on kernels:                   Yes
  Integrated GPU sharing Host Memory:           No
  Support host page-locked memory mapping:      Yes
  Alignment requirement for Surfaces:           Yes
  Device has ECC support:                      Disabled
  CUDA Device Driver Mode (TCC or WDDM):        WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:              Yes
  Device supports Compute Preemption:           Yes
  Supports Cooperative Kernel Launch:           Yes
  Supports MultiDevice Co-op Kernel Launch:     No
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.3, CUDA Runtime Version = 12.3, NumDevs = 1
Result = PASS
```

Figure 1: Output of *deviceQuery*

Output of *bandwidthTest* in */1-Utilities*:

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

Device 0: NVIDIA GeForce GTX 1060
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(GB/s)
  32000000                   12.0

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(GB/s)
  32000000                   10.8

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(GB/s)
  32000000                   131.1

Result = PASS
```

Figure 2: Output of *bandwidthTest*

*What is the Compute Capability of your GPU device?*

The compute capability of the NVIDIA GeForce GTX 1060 is: 6.1

*How will you calculate the GPU memory bandwidth (in GB/s) using the output from deviceQuery? (Hint: memory bandwidth is typically determined by clock rate and bus width, and check what double data rate (DDR) may impact the bandwidth). Are they consistent with your results from bandwidthTest?*

Memory Bandwidth = Memory Clock Speed (In Hz) \* Memory Bus Width (In bits) \* 2 (Because of double data rate) / (10<sup>9</sup> \* 8) (To convert to GB/s)

Based on the deviceQuery output, for the test GPU, this gives:

$$4004 * 10^6 \text{ Hz} * 192 \text{ bit} * 2 / (10^9 * 8) = 192.192 \text{ GB/s}$$

This is consistent with NVIDIA's own data sheet for the GTX 1060.

### Exercise 3

*Compile both OMP and CUDA versions of your selected benchmarks. Do you need to make any changes in Makefile?*

No changes to the Makefile were required, but for simplicity's sake we removed the targets for the benchmarks we did not plan on using.

*Ensure the same input problem is used for OMP and CUDA versions. Report and compare their execution time.*

We decided to compare the performance on benchmarks *bfs* and *streamcluster*. CUDA times include both memory copy, and kernel execution times. For both *bfs* and *streamcluster* using CUDA, approximately half of the total compute time is taken up by memory copy between the host and the device.

For *bfs*, the program did print compute time for the OpenMP version, but not for the CUDA version, so for that benchmark, we used nvprof to gather detailed GPU execution times. *streamcluster*, on the other hand, did print the total execution time of the program for both OpenMP and CUDA versions, so to make it a fair comparison, we used that time for both executions, instead of using nvprof.

Results are displayed in table 1

	CUDA	OpenMP
bfs	10.457 ms	101.347 ms
streamcluster	4556 ms	14106 ms

Table 1: Compute time CUDA vs OpenMP

*Do you observe expected speedup on GPU compared to CPU? Why or Why not?*

Both of the chosen benchmarks are highly parallelizable, since not much synchronization is needed to produce a correct solution, and thus benefit greatly from GPU acceleration. This is evident in the timing results.

## Exercise 4

*How do you launch the code on GPU on Dardel supercomputer?*

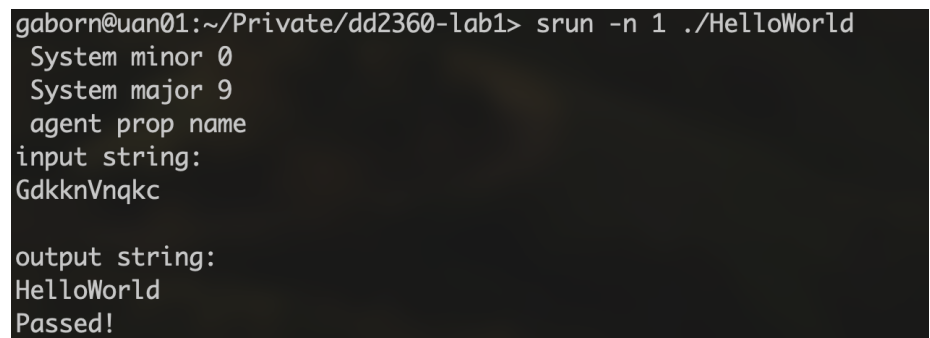
We uploaded the files to Dardel using SFTP, and then ran *make* to compile the project. After that, we requested a compute node:

```
salloc -A edu23.dd2360 -p gpu -N 1 -t 00:10:00
```

Finally, we ran the compiled executable on the allocated node:

```
srun -n 1 ./HelloWorld
```

*Output of **HelloWorld**:*



```
gaborn@uan01:~/Private/dd2360-lab1> srun -n 1 ./HelloWorld
System minor 0
System major 9
agent prop name
input string:
GdkknVnqkc

output string:
HelloWorld
Passed!
```

Figure 3: Output of HelloWorld