Applied GPU Programming - Assignment 1

Group 19 - Diogo Gaspar && Diogo Melita

Contributions:
We worked together, in person, so both of us had the same workload and worked on the same (all) exercises.

Diogo Melita - `diogogm@kth.se` - 50%
Diogo Gaspar - `dgaspar@kth.se` - 50%

Exercise 1 - Reflection on GPU-accelerated Computing

1. In terms of architecture, a CPU has tens of massive cores, with lots of hardware for control and less Arithmetic Logic Units (ALUs), exceling at irregular-control-intensive work. On the other hand, GPUs have thousands of small cores, lots of ALUs and less hardware for control, perfoming better at regular-math-intensive work.

2. From the June 2024 Top500 Supercomputers list (https://top500.org/lists/top500/2024/06/), 9 of the top 10 supercomputers use GPUs:

Rank	System	Processor	Vendor
1st	Frontier	AMD Instinct MI250X	AMD
2nd	Aurora	Intel Data Center GPU Max	Intel
3rd	Eagle	NVIDIA H100	NVIDIA
5th	LUMI	AMD Instinct MI250X	AMD
6th	Alps	NVIDIA GH200 Superchip	NVIDIA
7th	Leonardo	NVIDIA A100 SXM4	NVIDIA
8th	MareNostrum 5 ACC	NVIDIA H100	NVIDIA
9th	Summit	Volta GV100	NVIDIA
10th	Eos NVIDIA DGX SuperPOD	NVIDIA H100	NVIDIA

The Fugaku Supercomputer (ranked 4th) is the only one without a GPU.

We found it interesting that a couple of GPUs show up more than once: AMD's Instinct M250X, as well as NVIDIA's H100, both show up thrice in this list.

3. $PowerEfficiency(Flops/W) = \frac{Rmax \times 10^{15}}{Power \times 10^{3}}$

- Frontier:
  $PowerEfficiency = \frac{1206 \times 10^{15}}{22786 \times 10^{3}} = 5.293 \times 10^{10}\,\mathrm{FLOPS/W}$
- Aurora
  $PowerEfficiency = \frac{1012 \times 10^{15}}{38698 \times 10^{3}} = 2.614 \times 10^{10}\,\mathrm{FLOPS/W}$
- Eagle
  We cannot calculate the Power Efficiency as the table did not provide Power Statistics, so we could not calculate the power efficiency.
- Fugaku
  $PowerEfficiency = \frac{442.01 \times 10^{15}}{29899 \times 10^{3}} = 1.478 \times 10^{10}\,\mathrm{FLOPS/W}$
- LUMI
  $PowerEfficiency = \frac{379.7 \times 10^{15}}{7101 \times 10^{3}} = 5.34 \times 10^{10}\,\mathrm{FLOPS/W}$
- Alps
  $PowerEfficiency = \frac{270 \times 10^{15}}{5194 \times 10^{3}} = 5.20 \times 10^{10}\,\mathrm{FLOPS/W}$
- Leonardo
  $PowerEfficiency = \frac{241.2 \times 10^{15}}{7494 \times 10^{3}} = 3.22 \times 10^{10}\,\mathrm{FLOPS/W}$
- MareNostrum 5 ACC
  $PowerEfficiency = \frac{175.3 \times 10^{15}}{4159 \times 10^{3}} = 4.21 \times 10^{10}\,\mathrm{FLOPS/W}$
- Summit
  $PowerEfficiency = \frac{148.6 \times 10^{15}}{10096 \times 10^{3}} = 1.471 \times 10^{10}\,\mathrm{FLOPS/W}$
- Eos NVIDIA DGX SuperPOD
  We cannot calculate the Power Efficiency as the table did not provide Power Statistics, so we could not calculate the power efficiency.

Exercise 2 - Query Nvidia GPU Compute Capability

1. The screenshot can be seen below:

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          12.2 / 12.2
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 15102 MBytes (15835660288 bytes)
  (040) Multiprocessors, (064) CUDA Cores/MP:    2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 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:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  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 3 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  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:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 4
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.2, CUDA Runtime Version = 12.2, NumDevs = 1
Result = PASS
```

2. As it can be seen in the output, under “CUDA Capability Major/Minor version number”, the compute capability of the device is 7.5.

3. The screenshot can be seen below:

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

  Device 0: Tesla T4
  Quick Mode

  Host to Device Bandwidth, 1 Device(s)
  PINNED Memory Transfers
    Transfer Size (Bytes)        Bandwidth(GB/s)
    32000000                     11.2

  Device to Host Bandwidth, 1 Device(s)
  PINNED Memory Transfers
    Transfer Size (Bytes)        Bandwidth(GB/s)
    32000000                     12.2

  Device to Device Bandwidth, 1 Device(s)
  PINNED Memory Transfers
    Transfer Size (Bytes)        Bandwidth(GB/s)
    32000000                     239.4

 Result = PASS

 NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

4.

$$\text{Memory Bandwidth (GB/s)} = \frac{\text{Memory Clock Rate (MHz)} \times \text{Memory Bus Width (bits)} \times \text{DDR factor}}{8 \times 10^9}$$

Having our output from 1. into account, we have the following:

- The Memory Clock Rate is 5001 MHz.
- The Memory Bus Width is 256 bits.

Furthermore, the DDR factor is 2 for GDDR memory, since we're dealing with double data rate.

Thus:

$$\text{Memory Bandwidth (GB/s)} = \frac{5001 \times 10^6 \times 256 \times 2}{8 \times 10^9} \approx 320\text{GB/s}$$

In our bandwidth test output in 3., the Device to Device Bandwidth is 239.4 GB/s, which is lower than the theoretical 312.5 GB/s calculated above. This difference, however, can be assumed to be normal as the calculations assume ideal conditions, whereas the actual test includes overheads and real-world inefficiencies, such as latency, bus contention, and memory controller limitations.

Exercise 3 - Rodinia CUDA benchmarks and Comparison with CPU

1. We changed the Makefiles so that, inside `cuda/cfd`, its Makefile had its `nvcc` calls updated, with `gpu-architecture` and `gpu-code` being changed from `compute_20` to `compute_75`, reflecting the aforementioned *CUDA Capability Major/Minor version number*, which is 7.5. Furthermore, we ended up only including the CUDA and OMP targets, since those were the only ones we were going to use, and from those, we only ran make for the benchmarks we opted for: `heartwall` and `lud`.

2. For `heartwall`, `run` with the GPU (CUDA) had an execution time of 0.463s, while with the CPU (OpenPM) we saw an execution time of 14.277s, resulting in a speedup of 30.836. `lud`'s case was even more severe: CUDA saw an execution time of 0.275s, while OpenPM oversaw one of 52.083s, which represents a speedup of 189.393!

3. As mentioned in 2., we see huge speedups in execution time for both benchmarks. In our opinion, this is due to the natural differences between GPU and CPU architectures: GPUs have thousands of small cores, which lead to them excelling in regular math-intensive work, as well as image-processing ones; this is in opposition to CPUs, which have tens of massive cores, leading to them excelling in other areas, in irregular control-intensive work.
   `heartwall` is a benchmark which "tracks the movement of a mouse heart over a sequence of 104 609×590 ultrasound images", while `lud` "is an algorithm to calculate the solutions of a set of linear equations". As such, and with our previous considerations in mind, it makes sense for us to see a huge speedup in both scenarios.

Exercise 4 - Run a Simple CUDA code on Nvidia GPU

1. We used the following Makefile, inspired by examples from the internet:

```
# Compiler and flags
NVCC = nvcc
NVCC_FLAGS = -arch=sm_52 -std=c++11

# Target and source files
TARGET = cuda_program
SRC = simple-1.cu

# Default rule to build the target
all: $(TARGET)

# Rule to compile and link the CUDA program
$(TARGET): $(SRC)
	$(NVCC) $(NVCC_FLAGS) -o $(TARGET) $(SRC)

# Clean rule to remove generated files
clean:
	rm -f $(TARGET)
```

2. From the program's output, we can gather that there are 4 blocks, and each one has 4 different thread IDs. As such, there are $4 \times 4 = 16$ threads launched by this code. We can confirm this by looking at the source code and noticing that we define a two-dimensional grid of 4 blocks, `dim3 dimGrid(2, 2)` and two-dimensional blocks of 4 threads each, `dim3 dimBlock(2, 2)`. Therefore, the total number of CUDA threads is the product of the grid size by the block size, $4 \times 4 = 16$ CUDA threads.