GPUs and Heterogeneous Systems – A.Y. 2023-24

Scuola di Ingegneria Industriale e dell'Informazione
Prof. Antonio Miele

POLITECNICO
MILANO 1863

June 21, 2024 - **FIRST PART OF THE EXAM**

Surname:	Name:	Personal Code:

Question	1	2	3	4	5	OVERALL
Max score	3	3	3	3	3	15
Score						

Instructions:

- This first part of the exam is "closed book". The students are not allowed to consult any course material and notes.
- No extra devices (e.g., phones, iPad) are allowed. Please, shut down and store any electronic device.
- Students are not allowed to communicate with any other ones.
- Students can write in pen or pencil, any color, but avoid writing in red.
- Any violation of the above rules will lead to the invalidation of the test.
- **Duration: 30 minutes**

Question 1

Explain why NVIDIA GPUs do not classically provide any hardware support for grid-level synchronization.

Grid-level synchronization would require hardware synchronization mechanisms among different streaming multiprocessors. They have not been added to keep the architecture as simple as possible to privilege the performance of this kind of throughput-oriented system.

Question 2

Write the formula for computing the arithmetic intensity of a kernel and apply it to the following function.

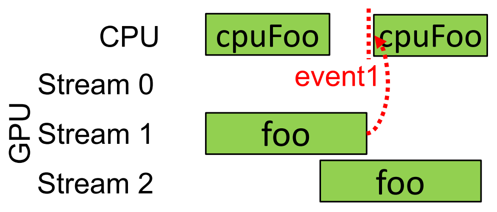
```
#define STRUCT_DIM 3

__global__ void foo(float *input, float *output){
  const int i = blockIdx.x*blockDim.x + threadIdx.x;
  const float a = input[i*STRUCT_DIM];
  const float b = input[i*STRUCT_DIM+1];
  const float c = input[i*STRUCT_DIM+3];
  output[i] = (b-a)/c + (c-b)/(a+b);
}
```

Arithmetic intensity: (#floating point operations) / #transfered bytes with memory

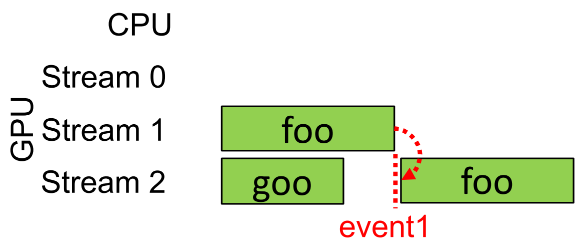
In the example:
#floating point operations = 6 (2 subtractions + 2 divisions + 2 sums)
#transfered bytes with memory = 16 (3 reads and 1 store; each float is 4 bytes)
Arithmetic intensity = 6/16

Question 3
Draw the Gantt chart of the execution of the various functions in the following three cases.

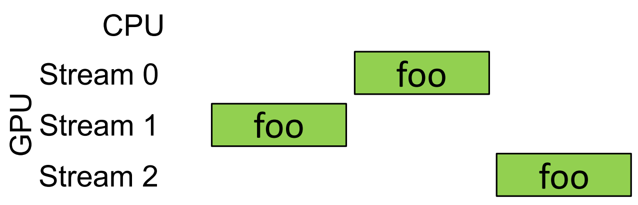
(a) Assume that `foo` execution is longer than `cpuFoo` one.
```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
foo<<<blocks, threads, 0, stream1>>>();
cudaEventRecord(event1, stream1);
cpuFoo();
foo<<<blocks, threads, 0, stream2>>>();
cudaEventSynchronize(event1);
cpuFoo();
```



(b) Assume that `foo` execution is longer than `goo` one.
```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
foo<<<blocks, threads, 0, stream1>>>();
cudaEventRecord(event1, stream1);
goo<<<blocks, threads, 0, stream2>>>();
cudaStreamWaitEvent(stream2, event1);
foo<<<blocks, threads, 0, stream2>>>();
```



(c)
```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
foo<<<blocks, threads, 0, stream1>>>();
foo<<<blocks, threads>>>();
foo<<<blocks, threads, 0, stream2>>>();
```



Question 4
Let's assume to run the following kernel on a Maxwell (or more recent) architecture and to size the grid with a single block of 32 threads; which is the efficiency of global load and store operations of the following CUDA kernel? Motivate the answer.

```
__global__ void foo(char* a, char* b){
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  b[(i+3)% blockDim.x] = a[(i*2)%blockDim.x];
}
```

Load efficiency = 50%. The access pattern to the read array is: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30. So, a single transaction is enough to serve all the requests, but one is not used for every two subsequent elements.
Store efficiency = 100%. The access pattern is: 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 1 2. So, a single transaction is enough to serve all the requests, and all elements are written.

Question 5
Explain why OpenCL mainly adopts the just-in-time (JIT) compilation for the kernels to execute and which is the main exception where the JIT approach cannot be used.

OpenCL adopts JIT compilation to offer high flexibility and transparency w.r.t. the considerable variety of heterogeneous devices that possibly may execute the kernel; JIT compilation enables runtime discovery of the actual underlying architecture where the code is being executed and runtime selection of the final target device where to offload the kernel. JIT compilation cannot be used when targeting FPGA devices because it requires a synthesis of the hardware accelerator that is not an “immediate” process.