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

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

POLITECNICO
MILANO 1863

July 19, 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** | | | | | | |

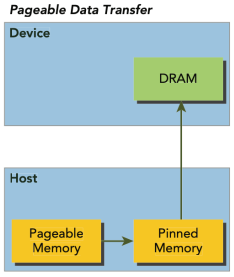
Instructions:

- 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**
Briefly explain what a shader program is.

In computer graphics, the shader program implements the logic of an elaboration related to one of the various steps in 3D rendering.

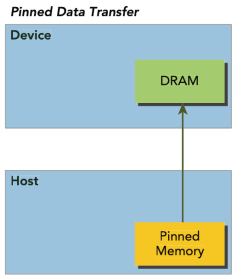
**Question 2**
Briefly explain how synchronous and asynchronous memory data transfers work in NVIDIA GPU.




In synchronous data transfer, the host program allocates pageable memory to store its data. When the `cudaMemcpy` function is called to send data from the host memory to the device one:

1. CUDA driver allocates page-locked (pinned) host memory.
2. CUDA driver copies data from the page-locked (pinned) host memory to the pinned memory.
3. GPU DMA transmits data from the pinned memory to the device one.

During this process, the host program gets blocked, waiting for the completion of the copy.




In asynchronous data transfer, the host program calls the `cudaMallocHost` function to allocate pinned memory and uses it to directly store data. When the `cudaMemCpyAsync` function is called, the GPU DMA is programmed to send data from the pinned memory to the device one. Since pinned memory is directly used from the beginning, the 2-hop copy process is not required and this allows the host program to not be blocked during the copy process.

**Question 3**

For each of the two following device characterizations, evaluate (and motivate) whether the kernel reported below is compute-bound or memory-bound:

1. Peak FLOPS=150 GFLOPS, peak memory bandwidth=100 GB/second
2. Peak FLOPS=200 GFLOPS, peak memory bandwidth=500 GB/second

```
__global__ void foo(float *in1, float *in2, float *in3, float *output){
  const int i = blockIdx.x*blockDim.x + threadIdx.x;
  const float a = in1[i];
  const float b = in2[i];
  const float c = in3[i];
  output[i] = (a+b)/c + (a+c)/b + (b+c)/a;
}
```

A kernel is compute-bound when: arithmetic intensity * peak memory bandwidth > Peak FLOPS. Otherwise it is memory-bound. Arithmetic intensity is computed as (#floating point operations) / #transfered bytes with memory

In the example:
#floating point operations = 8 (5 sums +3 divisions)
#transfered bytes with memory = 16 (3 reads and 1 store; each float is 4 bytes)
Arithmetic intensity = 8/16 = 0.5

Case 1: 0.5 * 100 = 50 < 150 -> the kernel is memory-bound
Case 2: 0.5 * 500 = 250 > 200 -> the kernel is compute-bound

**Question 4**

The two following kernels perform the same elaboration on a list of pairs of values. In the implementation on the left, a struct of arrays organization of the data is used, while in the implementation on the right, an array of structs organization. Let's assume to run the two kernels 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 in the two cases? Motivate the answer.

```
typedef struct {
  char x[N];
  char y[N];
} struct_of_arrays_t;

__global__ void foo(struct_of_arrays_t *data){
  const int i = blockIdx.x*blockDim.x +
                threadIdx.x;
  data->y[i] = data->x[i] * 2;
}
```

```
typedef struct {
  char x;
  char y;
} innerStruct_t;

typedef innerStruct_t array_of_structs_t[N];

__global__ void foo(array_of_structs_t *data){
  const int i = blockIdx.x*blockDim.x + threadIdx.x;
  data[i].y = data[i].x * 2;
}
```

In the case on the left, all `x` values are stored in consecutive 1-byte memory locations; therefore, the efficiency of global load operations is 100% since consecutive threads require the read of consecutive memory locations. In the case on the right, `x` values and `y` values are alternated in the memory; therefore, the efficiency of global load operations is 50% since consecutive threads require to read 1-byte memory locations that are separated by 1 byte of other content.vIn both cases, the same considerations can be drawn for store operations.

**Question 5**

Briefly explain the main advantages and drawbacks of OpenACC w.r.t. CUDA.

Advantages:

- High portability - The source code can be targeted to different accelerators by recompiling
- High programmability - The original source code is simply annotated with directives

Drawbacks:

- Lower performance than CUDA - Due to the support of multiple devices and the high level of abstraction, specific optimizations cannot be applied by means of annotations