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

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

POLITECNICO
MILANO 1863

January 28, 2025 - **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 tensor core is, which operation it performs, and why it has been added to the recent GPU architectures.

Due to the increasing trend in executing deep learning applications onto GPU, NVIDIA has integrated into the streaming multiprocessor a specific hardware module accelerating basic deep learning computations (particularly convolutions. The tensor core is actually a hardware module executing multiplication and accumulation operations onto 4x4 matrices of floating-point values; such an operation is the basic tile of a convolution.

Question 2
Explain the streaming multiprocessor occupancy index: formula, why it is relevant, and what it is used for.

The GPU is a throughput-oriented system; therefore, the best performance is achieved by maximizing resource utilization. In turn, high resource utilization is obtained by interleaving many warps in the streaming multiprocessor so that when a warp stalls, there is a high probability that another warp is ready to be executed, always keeping the streaming multiprocessor busy.
In this context, the maximization of the streaming multiprocessor occupancy, defined as $\text{SM occupancy} = \frac{\text{active warps}}{\text{maximum warps}}$, is the starting point in kernel performance optimization.

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

Indexes used by the 32 threads for reading the 1-byte long elements of vector a are:
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
Therefore, each position is accessed twice, a single block is read, and a byte every two in the block is used. In conclusion, the global load efficiency is 50%.

Indexes used by the 32 threads for writing the 1-byte long elements into vector b are:
1, 2, 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
Therefore, each thread accesses a contiguous 1-byte-long element. In conclusion, the global store efficiency is 100%.

Question 4
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=100 GFLOPS, peak memory bandwidth=250 GB/second
2. Peak FLOPS=150 GFLOPS, peak memory bandwidth=100 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-a-b)/c + (b-a-c)/c) * 3;
}
```

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 exercise:
#floating point operations = 12 (6 subtractions + 2 sums + 3 divisions + 1 multiplication)
#transfered bytes with memory = 16 (3 reads and 1 store; each float is 4 bytes)
Arithmetic intensity = 12/16 = 3/4 = 0.75

Case 1: 0.75 * 250 = 187.5 > 100 -> the kernel is compute-bound
Case 2: 0.75 * 100 = 75 < 150 -> the kernel is memory-bound

Question 5
Is it possible to parallelize the following snippet of code using OpenACC pragmas? Motivate the answer.

```
int sum=0;
for(int i = 0; i < N; i++)
  sum += A[i];
```

The loop accumulates on `sum` all the values contained in A. This can be implemented as a parallel reduction, supported by OpenACC by means of the following pragma:

```
#pragma acc parallel loop reduction(+:sum)
```