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

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

POLITECNICO MILANO 1863

January 10, 2025 - **FIRST PART OF THE EXAM**

Surname:	Name:	Personal Code:

Question	**1**	**2**	**3**	**4**	**5**	**6**	**OVERALL**
Max score	**3**	**3**	**3**	**3**	**3**	**3**	**18**
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: 40 minutes**

Question 1
Briefly explain what SGI RealityEngine is.

SGI RealityEngine (2013) is a pre-GPU era computer that accelerates the graphics pipeline for 3D rendering. It is programmable compliant with OpenGL and targets a limited market of large companies and universities aiming to perform computer simulation, digital content creation, engineering, and research.

Question 2
Specify in which NVIDIA GPU architecture the following mechanisms were introduced for the first time:

- Dynamic parallelism: Kepler
- Unified shader processor: Tesla
- Multi Instance GPU (MIG) virtualization: Ampere

Question 3
Simulate the following simple sum reduction kernel and show the `data` array's content at each loop iteration. Consider a grid with 2 blocks, each block with 4 threads; the initial content of the `data` array is:
`[0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]`

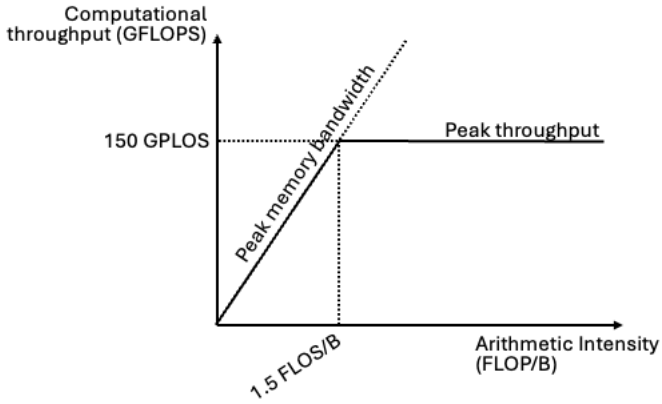
```
#define STRIDE_FACTOR 2

__global__ void reduce(double* data) {
  int i = threadIdx.x * STRIDE_FACTOR;
  int base_i = blockDim.x * blockIdx.x * STRIDE_FACTOR;
  for (int stride = 1; stride <= blockDim.x; stride *= STRIDE_FACTOR) {
    if (threadIdx.x % stride == 0) {
      data[base_i + i] += data[base_i + i + stride];
    }
    __syncthreads();
  }
}
```

```
Input:       [ 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Iteration 1: [ 1, 1, 5, 3, 1, 1, 5, 3, 1, 1, 5, 3, 1, 1, 5, 3]
Iteration 2: [ 6, 1, 5, 3, 6, 1, 5, 3, 6, 1, 5, 3, 6, 1, 5, 3]
Iteration 3: [12, 1, 5, 3, 6, 1, 5, 3,12, 1, 5, 3, 6, 1, 5, 3]
```

Question 4

Draw the roofline model for a computing system having the following characteristics:

- Peak computational throughput = 150 GFLOPS
- Peak memory bandwidth = 100 GB/second



The x coordinate for the intersection between the two lines representing the peak memory bandwidth, and the peak computational throughput can be computed as:

Arithmetic intensity = peak throughput / peak memory bandwidth = (150 GFLOPS) / (100 GB/s) = 1.5 FLOP/B

Question 5

Briefly explain what this piece of OpenCL code will print on the screen.

```
/* For the exercise we assume no error may occur. */
/* Moreover, macro values are not relevant */

#include "CL/cl.h"
#include <stdio.h>
#define MAXPLATFORMS ...
#define MAXDEVICES ...
#define MAXSTRING ...

int main(){
  int i, j;
  char text[MAXSTRING];
  cl_platform_id platformIds[MAXPLATFORMS];
  cl_device_id deviceIds[MAXDEVICES];
  cl_uint numPlatforms, numDevices;

  clGetPlatformIDs(0, NULL, &numPlatforms);
  clGetPlatformIDs(numPlatforms, platformIds, NULL);
  for (i=0; i<numPlatforms; i++){
    clGetPlatformInfo(platformIds[i], CL_PLATFORM_NAME, MAXSTRING, text, NULL);
    clGetDeviceIDs(platformIds[i], CL_DEVICE_TYPE_GPU, 0, NULL, &numDevices);
    if(numDevices>0){
     printf("%s :\n", text);
     clGetDeviceIDs(platformIds[i], CL_DEVICE_TYPE_GPU, numDevices, deviceIds, NULL);
      for (j=0; j<numDevices; j++){
          clGetDeviceInfo(deviceIds[j], CL_DEVICE_NAME, MAXSTRING, text, NULL);
        printf("%s\n", text);
      }
    }
  }
  return 0;
}
```

For each OpenCL platform, the program checks if it contains at least 1 GPU device. If so, the program prints the name of the platform and the related GPU devices.

Question 6

In the following snippet of code using OpenACC pragmas, how many times will `foo()` and `bar()` be executed? Motivate the answer.

```
#pragma acc parallel num_gangs(16)
{
  #pragma acc loop gang
  for (int i=0; i<n; i++) {
    bar(i);
  }
  foo();
}
```

The first pragma generates 16 gangs, each with a single worker and vector element, to execute the code between the brackets. Then, the second pragma uses the generated gangs to parallelize the subsequent loop. The code after the loop (`foo` call) is not affected by the second pragma, and therefore it is executed in a parallel redundant way. Therefore, `bar` is called `n` times and then `foo` is called 16 times.