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.

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

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;
}
```

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;
}
```

Question 5
Briefly explain the main advantages and drawbacks of OpenACC w.r.t. CUDA.

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

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

POLITECNICO
MILANO 1863

July 19, 2024 - **SECOND PART OF THE EXAM**

Surname:	Name:	Personal Code:

Question	1	2	3	OVERALL
Max score	5.5	5.5	5	16
Score				

Instructions:

- This second part of the exam is "open book". The students are allowed to use any material and notes.
- The students are allowed to use the laptop and the tablet. No extra devices (e.g., phones) are allowed. Please, shut down and store not allowed electronic devices.
- Students are not allowed to communicate with any other one or use Internet.
- Students can write in pen or pencil, any color, but avoid writing in red.
- Students can also use the laptop to code the test solution. In this case, please pay attention to the instructor's instructions to submit the test solution.
- Any violation of the above rules will lead to the invalidation of the test.
- **Duration: 1 hour and 15 minutes**

Question 1
Implement simple CUDA kernel functions to accelerate the compute-intensive functions (`mult` and `compare`) in the following C program.

Question 2
Modify the main function to execute the CUDA kernel function defined in the former question. Set the block size to 32 for the first function (`mult`) and to 32x32 for the second one (`compare`).

Question 3
Implement a new CUDA kernel function to accelerate the second compute-intensive function (`compare`) in the following C program using the shared memory.

The source code can be downloaded from the course page on WeBeep

```c
/*
 * The kernel function 1 (mult) performs the multiplication of a vector by a scalar value.
 * The kernel function 2 (compare) receives two vectors of integers, called A and B,
 * together with the sizes sa and sb, and a third empty vector of integers, C, which
 * size is sa*sb.
 * For each pair A[i] and B[j], the function saves in C[i][j] value 1 if A[i] > B[j],
 * 0 otherwise (do consider that the function manages C as a linearized array).
 * The main function is a dummy program receiving in input sa and sb, populating randomly A
 * and B, invoking the above two functions and showing results.
 */

#include <stdio.h>
#include <stdlib.h>

#define VALUE 10

void compare(int *M, int *N, int dm, int dn, int *P);
void mult(int *V, int dim, int fatt, int *P);

//kernel function 1: vector per scalar multiplication
void mult(int *V, int dim, int fatt, int *P){
  int i;
  for(i=0; i<dim; i++)
    P[i] = V[i] * fatt;
}

//kernel function 2: compare each element of M against any element of N
void compare(int *M, int *N, int dm, int dn, int *P){
  int i, j;
  for(i=0; i<dm; i++)
    for(j=0; j<dn; j++)
      P[i * dn + j] = (M[i] > N[j]);
}

int main(int argc, char **argv) {
  int *A, *B, *A1, *B1, *C;
  int sa, sb;
  int i, j;

  //initialize sa and sb
  //...

  //allocate memory for the three vectors
  A = (int*) malloc(sizeof(int) * sa);
  B = (int*) malloc(sizeof(int) * sb);
  A1 = (int*) malloc(sizeof(int) * sa);
  B1 = (int*) malloc(sizeof(int) * sb);
  C = (int*) malloc(sizeof(int) * sa*sb);
  //check if memory is correctly allocated
  //...

  //initialize input vectors A and B
  //...

  //execute on CPU
  mult(A, sa, VALUE, A1);
  mult(B, sb, VALUE, B1);
  compare(A1, B1, sa, sb, C);

  //print results
  //...

  free(A);
  free(B);
  free(A1);
  free(B1);
  free(C);

  return 0;
}
```