GPU and Heterogeneous Systems – A.Y. 2021-22

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

July 15, 2022 – **FIRST PART OF THE EXAM**

Surname:	Name:	Person Code:

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

Instructions:

- **Duration: 40 minutes**
- This first part of the exam is "closed book". The students are not allowed to consult any 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.

Question 1
Explain the peculiarities of the SIMT approach adopted in NVIDIA GPUs w.r.t. the classical vector processors.

Question 2
Complete the following OpenACC pragmas to optimize data transfer between the host and the device. In particular, 1) specify between brackets () the appropriate list of arrays (index range for each array is always `[0:N]`), and 2) delete unnecessary clauses. Motivate your answer.

Used as input in both loops

Local variable used in both loops

```
void foo(int *A, int *B, int *D){
  int C[N];
#pragma acc data copy(      ) copyin(A[0:n]) copyout(      ) create(C[0:n])
{
  #pragma acc parallel loop copy(     ) copyin(B[0:n]) copyout(     ) create(     )
  for(int i = 0; i < N; i++)
    C[i] = A[i] + B[i];
  #pragma acc parallel loop copy(     ) copyin(     ) copyout(D[0:n]) create(     )
  for(int i = 0; i < N; i++)
    D[i] = C[i] + A[i];
}
}
```

The copy command copy from host memory to device memory and free this space at end of the program portion. So if done in the inner portions it has to be done twice, so slower program. Better do it one time at the beginning.

Question 3
Explain what pinned memory is and why it is necessary when using asynchronous data transfer.

gpu can't access pageable memory like the host memory, so we have first to pin a memory section in a fixed memory and then procede with the kernels launch. Moreover the host mem copy is a blocking process, instead using pinned we can share

Question 4
Which is the efficiency of global load and store operations of the following CUDA kernel? Assume to have a block of 32 threads and to run the application on a Maxwell architecture where L1 cache has 32-byte access width. Motivate your 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];
}
```

Store: accesses [2,3,4,...,31]
one transaction, Eff. 100%.
Load: access[0,2...,62,0,2...,62]
two transactions, Eff: 50%.

Question 5
Explain which is the bottleneck in the performance of the basic histogram algorithm and how it can be solved.

GPU and Heterogeneous Systems – A.Y. 2021-22

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

July 15, 2022 – **SECOND PART OF THE EXAM**

Surname:	Name:	Personal Code:

Question	1	2	3	OVERALL
Max score	5	5	6	16
Score				

Instructions:

- **Duration: 1 hour and 20 minutes**
- 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.

Question 1
Implement a basic CUDA kernel function to accelerate the compute-intensive function 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 each used dimension).

Question 3
Implement a new CUDA kernel function to accelerate the compute-intensive function in the following C program by using dynamic parallelism. Specify any change (possibly) applied to the main function.

The source code can be downloaded from: XXX

```
/*
* The following program elaborates a 2D matrix of integers where each row contains a
* variable number of elements. In particular, the kernel function receives in input
* the matrix and sum to each element a value obtained by multiplying the row number
* by the number of elements in the row.
* For instance, if we consider the following matrix with 4 rows:
* 1 2 3 4
* 1 2
* 1 1 1 1 1
* 2
* The kernel function will compute a new matrix as follows:
* 1 2 3 4
* 3 4
* 11 11 11 11 11
* 5
*
* The described matrix is represented in the program as follows:
* - An integer array A contains all the elements of the matrix in a linearized way
* - An integer variable NUMOFELEMS containing the overall number of elements in the matrix
* - An integer variable ROWS contains the number of rows in the matrix
* - An integer array COLOFFSETS contains the indexes in A where each matrix row starts
* - An integer array COLS contains the length of each row of the matrix
*
* Thus, the matrix above is modeled in the program as:
* A = [1 2 3 4 1 2 1 1 1 1 1 2]
* NUMOFELEMS = 12
* ROWS = 4
* COLOFFSETS = [0, 4, 6, 11]
* COLS = [4, 2, 5, 1]
*/

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

void printM(int* a, int rows, int* colOffsets, int *cols);

//kernel functions: sum to each element of the matrix a coefficient obtained
//by multiplying the row number by the row index
void elaborateMatrix(int* a, int rows, int* colOffsets, int *cols, int* b){
  int i, j, coeff;
  for(i=0; i<rows; i++){
    coeff = i*cols[i];
    for(j=0; j<cols[i]; j++)
      *(b + colOffsets[i] + j) = *(a + colOffsets[i] + j) + coeff;
  }
}
```

we can exploit parallelism in the second for, writing in the for the launch of a child kernel which will make the computation. So we'll have cols[i] child kernel which will work in parallel, for every i.

```
int main(int argc, char *argv[]){
  int *a, *b; //input and output matrices
  int rows; //number of rows
  int *cols; //contains the number of columns per each row
  int *colOffsets; //contains the indexes in the data array where each matrix row starts
  int numOfElems; //overall number of elements
  int i, j;

  //generate the matrix

  //call the kernel function
  elaborateMatrix(a, rows, colOffsets, cols, b);

  //print results

  //release memory
  free(a);
  free(b);
  free(cols);
  free(colOffsets);

  return 0;
}
```