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

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

January 9, 2023 – **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 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.

Question 1
Explain what a tensor core introduced in the Volta architecture is.

Question 2
Which is the efficiency of global load and store operations of the following CUDA kernel? Assume to have a block of 32 threads and run the code on a Maxwell architecture where L1 cache has 32-byte access width. Motivate the answer.

Store: 100%. one transaction [1,...,31,0]
Load: 3%. i · blockDim.x % blockDim.x = 0
Sempre (∀ i) → 32 transactions

```
__global__ void foo(char* a, char* b){
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  b[(i+1)%blockDim.x] = a[(i*blockDim.x)%blockDim.x];
}
```

Question 3
Briefly explain the main advantages and drawbacks of OpenACC w.r.t. CUDA.

Question 4
Describe the bottleneck in the CUDA-based acceleration of the Monte Carlo simulation algorithm and which strategy can be adopted to solve it.

Question 5
What will the following program fragment print on the screen? How many threads are created?

```
__global__ void foo(int size, int depth) {
  if (depth > 0) {
    if(threadIdx.x == 0){
      foo<<<1, size>>>(size, depth-1);
      cudaDeviceSynchronize();
    }
    __syncthreads();
  }
  printf("Depth: %d thread: %d block: %d\n", depth, threadIdx.x, blockIdx.x);
}

int main(){
  /*...*/
  int size = 4, depth = 2;
  foo<<<1, size>>>(size, depth);
  /*...*/
}
```

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

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

January 9, 2023 – **SECOND PART OF THE EXAM**

Surname:	Name:	Personal Code:

Question	**1**	**2**	**3**	**OVERALL**
Max score	**5**	**5,5**	**5,5**	**16**
Score				

Instructions:

- **Duration: 1 hour and 15 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 CUDA kernel function to initialize the data used in the compute-intensive portion in the following C program.

Question 2
Implement a CUDA kernel function to accelerate the compute-intensive portion in the following C program.

Question 2
Modify the main function to execute the two CUDA kernel functions defined in the former questions. Set block size to 32 (for each used dimension).

The source code can be downloaded from: https://miele.faculty.polimi.it/heatequation.c

```
/*
* The program simulates the heat transfer in a 2D surface. The 2D surface is modeled by
* means of a 2D array called surf (linearized for the sake of simplicity); each position in
* the array surf[i][j] contains the temperature of a single portion of the surface.
* Heat transfer is simulated by a partial differential equation (not discussed here in
* details...). The equation is solved by iterating over nsteps subsequent time steps; at
* each time step n, the new temperature of each position is computed as a function of the
* current temperatures of the same position surf[i][j] and of the 4 neighbor positions
* surf[i-1][j], surf[i][j-1], surf[i+1][j] and surf[i][j+1]; results are saved in a new
* array surf1, which is then swapped with surf to simulate subsequent time step.
* When computing the temperature of a position on the border of the surface, the
* non-existing neighbor positions are replaced with the temperature of the central position
* itself (i.e., when computing the temperature surf1[0][2], the neighbor position
* surf[-1][2] is replaced with surf[0][2]).
*/

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

#define A 0.5f    // Diffusion constant
#define DX 0.01f  // Horizontal grid spacing
#define DY 0.01f  // Vertical grid spacing
#define DX2 (DX*DX)
#define DY2 (DY*DY)
#define DT (DX2 * DY2 / (2.0f * A * (DX2 + DY2))) // Largest stable time step

int getIndex(int i, int j, int width);

int main(int argc, char **argv) {
  float *surf, *surf1, *tmp;
  int i, j, nx, ny, nsteps, n;

  // Read arguments
  if(argc != 4){
    printf("Please specify sizes of the 2D surface and the number of time steps\n");
    return 0;
  }
  nx = atoi(argv[1]);
  ny = atoi(argv[2]);
  nsteps = atoi(argv[3]);

  // Allocate memory for the two arrays
  surf = (float*) malloc(sizeof(float) * nx * ny);
  if(!surf){
    printf("Error: malloc failed\n");
    return 1;
  }
  surf1 = (float*) malloc(sizeof(float) * nx * ny);
  if(!surf1){
    printf("Error: malloc failed\n");
    return 1;
  }

  // Initialize the data with a pattern of disk of radius of 1/6 of the width
  float radius2 = (nx/6.0f) * (nx/6.0f);
  for (i = 0; i < nx; i++) {
    for (j = 0; j < ny; j++) {
      int index = getIndex(i, j, ny);
      // Distance of point i, j from the origin
      float ds2 = (i - nx/2) * (i - nx/2) + (j - ny/2)*(j - ny/2);
      if (ds2 < radius2)
        surf[index] = 65.0f;
      else
        surf[index] = 5.0f;
    }
  }
```

```
  // Compute-intensive kernel for the heat simulation
  int index;
  float sij, sim1j, sijm1, sip1j, sijp1;

  // Simulate N time steps
  for (n = 0; n < nsteps; n++) {
    // Go through the entire 2D surface
    for (i = 0; i < nx; i++) {
      for (int j = 0; j < ny; j++) {
        // Compute the heat transfer taking old temperature from surf
        // and saving new temperature in surf1
        index = getIndex(i, j, ny);
        sij = surf[index];
        if(i>0)
          sim1j = surf[getIndex(i-1, j, ny)];
        else
          sim1j = sij;
        if(j>0)
          sijm1 = surf[getIndex(i, j-1, ny)];
        else
          sijm1 = sij;
        if(i<nx-1)
          sip1j = surf[getIndex(i+1, j, ny)];
        else
          sip1j = sij;
        if(j<ny-1)
          sijp1 = surf[getIndex(i, j+1, ny)];
        else
          sijp1 = sij;
        surf1[index] = sij + A * DT *
            ( (sim1j - 2.0*sij + sip1j)/DX2 + (sijm1 - 2.0*sij + sijp1)/DY2 );
      }
    }
    // Swap the surf and surf1 pointers for the next time step
    // (it avoids copying all data from surf1 to surf)
    tmp = surf;
    surf = surf1;
    surf1 = tmp;
  }

  // Print results
  /* ... */

  // Free memory
  free(surf);
  free(surf1);

  return 0;
}

// Take in input 2D coordinates of a point in a matrix
// and translate in a 1D offset for the linearized matrix
int getIndex(int i, int j, int width) {
  return (i*width + j);
}
```