GPU programming – Lab 1 15.10.21 – A.A. 2021/22 Prof. L. Sterpone

Goal: Setting up the NVIDIA Jetson Nano kit and write your first cuda applications

Exercise 0 (setting up the board):

Please follow the instructions reported here:

https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#setup-display.

Tutorial 1 (the first cuda applications):

```
#include <stdio.h>
int main(void)
{
      printf("Hello World from Jetson CPU!\n");
}
```

1. Create a source file named *.cu
2. Compile the code using the nvcc compiler
3. Execute the code

```
#include <stdio.h>
__global__ void helloFromGPU (void) {
      printf("Hello World from Jetson GPU!\n");
}
int main(void) {
// hello from GPU
      printf("Hello World from CPU!\n");
      helloFromGPU <<<1, 10>>>();
      cudaDeviceReset();
      return 0;
}
```

1. Create a new source file named *.cu
2. Compile the code using the nvcc compiler
3. Execute the code

Exercise 1 (vector Add): The overall structure of a CUDA program that uses the GPU for computation is as follows:

1. Define the code that will run on the device in a separate function, called the kernel function.
2. In the main program running on the host's CPU:
    a. allocate memory on the host for the data arrays.
    b. initialize the data arrays in the host's memory.
    c. allocate separate memory on the GPU device for the data arrays.
    d. copy data arrays from the host memory to the GPU device memory.
3. On the GPU device, execute the kernel function that computes new data values given the original arrays. Specify how many blocks and threads per block to use for this computation.
4. After the kernel function completes, copy the computed values from the GPU device memory back to the host's memory.

Create a *vector_add* application performing the sum of two vectors of 16 elements on the Jetson GPU. Please consider the following steps:

Note

```
// The reference to the thread and block indexes
int tid = blockDim.x * blockIdx.x + threadIdx.x;
```

Step 1

```
//Creation of the memory pointers
int *a, *b, *c;               // The arrays on the host CPU machine
int *dev_a, *dev_b, *dev_c;   // The arrays for the GPU device
```

Step 2

```
//Allocate the memory on the CPU
a = (int*)malloc( N * sizeof(int) );
b = (int*)malloc( N * sizeof(int) );
c = (int*)malloc( N * sizeof(int) );
```

Step 3

```
//Initialize the vector on the CPU
```

Step 4

```
//Allocate the memory on the GPU
cudaMalloc( (void**)&dev_a, N * sizeof(int) );
cudaMalloc( (void**)&dev_b, N * sizeof(int) );
cudaMalloc( (void**)&dev_c, N * sizeof(int) );
```

Step 5

```
//Copy the arrays 'a' and 'b' to the GPU
cudaMemcpy( dev_a, a, N * sizeof(int),cudaMemcpyHostToDevice );
cudaMemcpy( dev_b, b, N * sizeof(int),cudaMemcpyHostToDevice );
```

Step 6

```
//Execute the vector addition on the GPU device,
```

Step 7

```
//Copy the array 'c' back from the GPU to the CPU
cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost );
```

Step 8

```
// verify that the GPU did the work we requested
```

Step 9

```
// free the memory we allocated on the CPU
free( a );
free( b );
free( c );
```

Step 10

```
// free the memory we allocated on the GPU
cudaFree( dev_a );
cudaFree( dev_b );
cudaFree( dev_c );
```