


AGH University of Science and Technology		
Introduction to CUDA and OpenCL.		Year: 2019/2020
Exercise no: 3	Subject: Work using Memory Menaged Utilities.	
Name and surname: Natalia Pluta Przemysław Grabowski		Team no: 02
Date of the laboratory: 14.10.2019 r	Date of submission: 28.10.2019 r	Note:

INTRODUCTION

In this lab, the aim was to manage the optimization of code with functions related to the allocation of memory and monitoring its status.

TASK 1.

- How large data structures are handled.

One more time we were working with the program called vectorAdd (from CUDA samples). It originally was adding two vectors of 50 000 elements each of float type. Every float element takes 4 bytes. To sum up it gives $50\,000 * 3 * 4 = 0.6 * 10^6$ bytes = 0.6 MB.

```
[[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

Picture 1: Original vectorAdd output (successful).

During the laboratory class, we had to find out the maximum number of elements in a vector. It turned out, by trial and error method, that if we chose the number of elements around $2 \ll 29 = 2^{30}$ (bitwise left shift) the program would not end up with success. By doing our calculations again we got $2^{30} * 3 * 4 \approx 12885$ MBytes. The total amount of global memory is equal to 6076 MBytes, we knew that thanks to the program called deviceQuery

that we got familiar with during the previous laboratory class. Well as we can see 12885 MB >> 6076 MBytes so the error occurs because we definitely run out of resources.

```
[Vector addition of 1073741824 elements]
Failed to allocate device vector B (error code out of memory)!
```

Picture 2: Output of vectorAdd with 2^{30} vector elements (unsuccessful).

As may be noted from the picture above, only vector A was allocated properly, but neither vector B nor vector C was given required amount of memory.

```
[Vector addition of 536870912 elements]
Failed to allocate device vector C (error code out of memory)!
```

Picture 3: Output of vectorAdd with 2^{29} vector elements (unsuccessful either).

When we downsized the number of elements to 2^{29} , we again received unsuccessful output, but this time only vector C was allocated improperly. We did our calculation once again $2^{29} * 3 * 4 \approx 6442$ MBytes, which is still more than available memory.

```
[Vector addition of 268435456 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 1048576 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

Picture 4: Output of vectorAdd with 2^{28} vector elements (successful).

Subsequently, we set the number of elements as 2^{28} after that operation vectors were finally allocated properly. It happened because our elements took $2^{28} * 3 * 4 \approx 3221$ MBytes of memory. We were no longer running out of memory since allocating vectors was not exceeding given resources.

TASK 2.

- How do we use Memory Managed Utilities?

In this lab, as a utility to manage memory, we used function `cudaMallocManaged()`. It allows to allocate memory for data that can be read or written from code running on either CPUs or GPUs. This function replaces `malloc()` or `new`, and is called in the same way. When code running on a CPU or GPU accesses data allocated this way the CUDA system software and/or the hardware takes care of migrating memory pages to the memory of the accessing processor.

The link to our code using function `cudaMallocManaged()`:

https://github.com/Myslav488/CUDA_labs/blob/master/vectorAdd_modified.cu

TASK 3.

- How to protect against too large data structure to be copied to GPU.

The overloading of the data in GPU is a serious problem, which we have to handle with. The solution for this is to check what maximum size of data there can be allocated in deviceQuery and then use that knowledge when typing the code. To prevent exceeding of the accessible memory we verify every time if our data is not bigger than we are able to allocate.