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: 03
Date of laboratory: 21.10.2019 r	Date of submission: 13.11.2019 r	Note:

INTRODUCTION

In this lab, we were analyzing the difference in the order of accessing unified memory to GPU and CPU and how it impacts on performance time.

TASK 1.

We had to conduct experiments to learn more about the behavior of 'cudaMallocManaged'. We had to check the following questions:

- What happens when unified memory is accessed only by the GPU?
- What happens when unified memory is accessed only by the CPU?
- What happens when unified memory is accessed first by the GPU then the CPU?
- What happens when unified memory is accessed first by the CPU then the GPU?

As the Unified Memory is shared for both GPU and CPU to accelerate computations, accessing UM for only one of GPU or CPU may cause more faults than with combining them.

Effect of execution proper codes on profiling Unified Memory is linked in the table below.

Unified Memory accessed:	-only by the GPU	-only by the CPU	-first by the GPU then the CPU	-first by the CPU then the GPU
Total CPU Page faults:	1536	1535	384	32

As seen above there is a slight difference between UM being accessed to only GPU or CPU. But both of those options are far more fault-causing than those with both CPU and GPU. Moreover, there is a significant prevalence when first accessed is CPU. It may be due to CPU is more suitable for serial instruction processing.

Page faults are misaccessings of a page by the hardware for software, when the page is mapped in the virtual address space, but not loaded in physical memory.

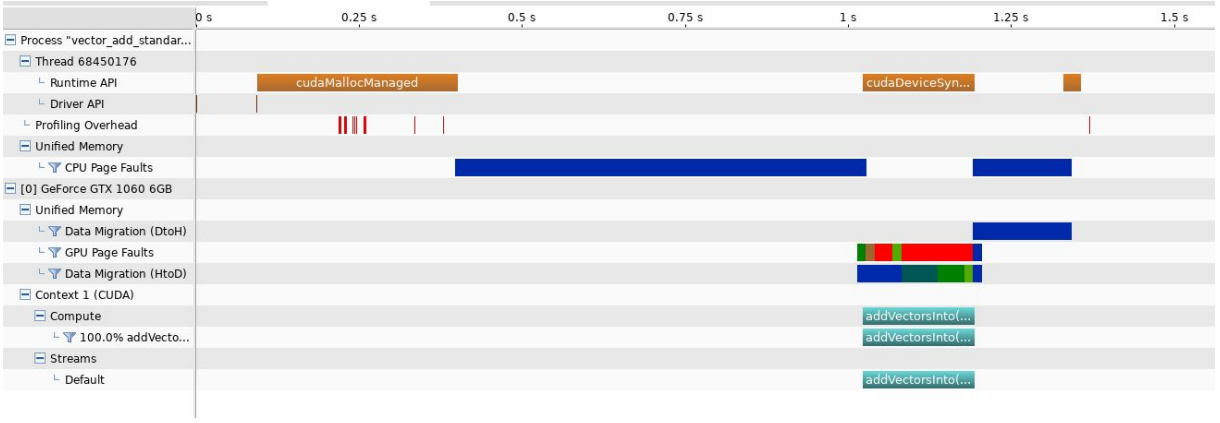
TASK 2.
Memory management analysis with nvvp and nvprof.

Nvvp is a very powerful graphical tool that has a lot of functionality, but during the laboratory class, we were focused on performance analyzing. This program gives us an opportunity to get a line on our application's CPU and GPU activity.

On the other hand, we have a program called **nvprof,** which is also a profiling tool. It enables us to collect and view data (such as GPU activities or CPU page faults) in Linux terminal.

We were studying various timelines of CUDA vector addition applications. Disproportions in time shown below are dependent on different types of data prefetching.

At first, we ran standard vector addition without any data prefetching. We also decided to profile this code by using nvvp and nvprof. As we can see in the pic. 1 and pic. 2 process called vector_add_standard lasted around 1.35 seconds, there was only one GPU activity - function addVectorsInto, which lasted around 152 milliseconds. The longest-running API calls were cudaMallocManaged and cudaDeviceSynchronize, altogether they took around 475 milliseconds, which is more than 95% time of API calls. They were 1536 CPU page faults (pic. 2), interestingly enough they lasted relatively long (pic.1), which caused an unnecessary extension of operating time. Moreover, the result of Unified Memory profiling can be observed at pic. 1 and pic. 2, from there we know that data migration (Host to Device and also Device to Host) lasted three times less than GPU page fault groups.



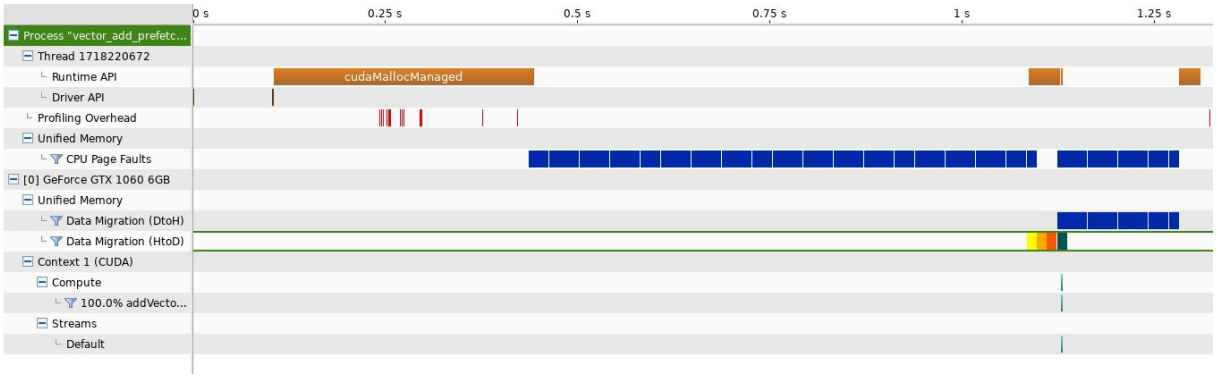
Pic. 1. The window of Nvidia Visual Profiler running vector_add_standard which stands for unified memory is being accessed only by the CPU.

```
[cuda-s18@lhcbgpu1 memory_management_lab]$ nvprof ./vector_add_standard
==38106== NVPROF is profiling process 38106, command: ./vector_add_standard
Success! All values calculated correctly.
==38106== Profiling application: ./vector_add_standard
==38106== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  151.99ms         1  151.99ms  151.99ms  151.99ms  addVectorsInto(float*,
float*, float*, int)
      API calls:   64.80%  322.51ms         3  107.50ms  22.070us  322.43ms  cudaMallocManaged
                   30.55%  152.04ms         1  152.04ms  152.04ms  152.04ms  cudaDeviceSynchronize
                    4.46%  22.197ms         3  7.3989ms  6.6229ms  7.8437ms  cudaFree
                    0.09%  455.23us         1  455.23us  455.23us  455.23us  cuDeviceTotalMem
                    0.06%  318.12us        96  3.3130us     838ns  115.17us  cuDeviceGetAttribute
                    0.02%  118.31us         1  118.31us  118.31us  118.31us  cudaLaunchKernel
                    0.01%  46.724us         1  46.724us  46.724us  46.724us  cuDeviceGetName
                    0.00%  15.155us         1  15.155us  15.155us  15.155us  cudaGetDevice
                    0.00%  9.6380us         1  9.6380us  9.6380us  9.6380us  cuDeviceGetPCIBusId
                    0.00%  5.5190us         3  1.8390us     978ns  3.4230us  cuDeviceGetCount
                    0.00%  4.0500us         1  4.0500us  4.0500us  4.0500us  cudaDeviceGetAttribute
                    0.00%  2.7930us         2  1.3960us  1.0470us  1.7460us  cuDeviceGet
                    0.00%  1.2580us         1  1.2580us  1.2580us  1.2580us  cuDeviceGetUuid
                    0.00%     839ns         1     839ns     839ns     839ns  cudaGetLastError

==38106== Unified Memory profiling result:
Device "GeForce GTX 1060 6GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   13763  28.512KB  4.0000KB  192.00KB  383.2148MB  44.27754ms  Host To Device
     766  171.03KB  4.0000KB  0.9961MB  127.9375MB  10.76186ms  Device To Host
    1123         -         -         -           -  149.0667ms  Gpu page fault groups
Total CPU Page faults: 1536
```

Pic. 2. The result of running vector_add_standard with nvprof, which stands for unified memory is being accessed only by the CPU.

Our primary goal was reducing the run-time of our program, so we tried to achieve that by prefetching data on GPU. As we can see in the pic. 3 and pic. 4, by this operation, we slightly improved our performance. The biggest difference is seen in GPU activities because addVectorsInto lasted only 2.5 milliseconds, which is 60 times less than it lasted last time (in vector_add_standard). And what is more, they were no longer any GPU page faults reported. Unfortunately, as we expected a number of total CPU Page faults remained the same.



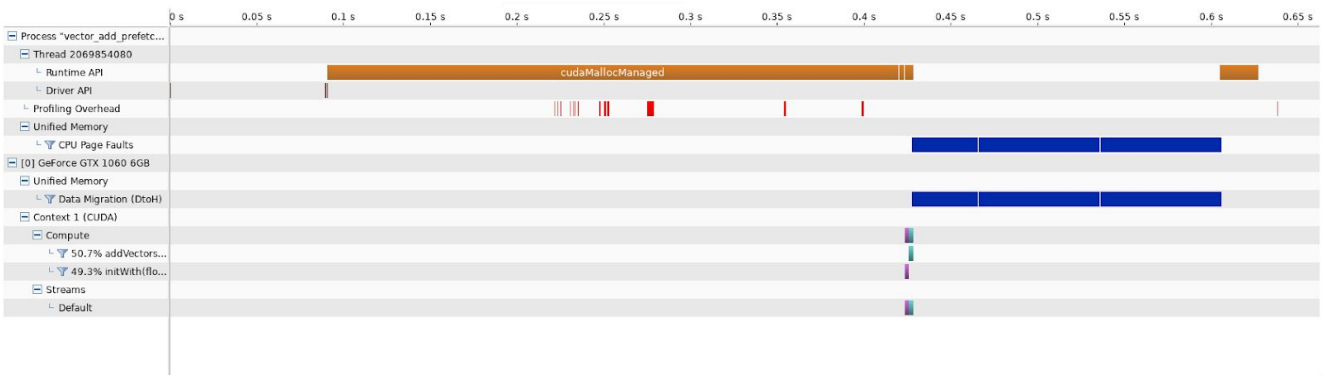
Pic. 3. A window of Nvidia Visual Profiler running vector_add_prefetch_gpu which stands for unified memory is being accessed only by the GPU.

```
[cuda-s18@lhcbgpu1 memory_management_lab]$ nvprof ./vector_add_prefetch_gpu
==36139== NVPROF is profiling process 36139, command: ./vector_add_prefetch_gpu
Success! All values calculated correctly.
==36139== Profiling application: ./vector_add_prefetch_gpu
==36139== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  2.5607ms         1  2.5607ms  2.5607ms  2.5607ms  addVectorsInto(float*,
float*, float*, int)
      API calls:   77.49%  312.58ms         3  104.19ms  53.080us  312.44ms  cudaMallocManaged
                   11.60%  46.809ms         1  46.809ms  46.809ms  46.809ms  cudaDeviceSynchronize
                    6.76%  27.264ms         3  9.0881ms  8.0469ms  10.287ms  cudaFree
                    3.84%  15.500ms         3  5.1668ms  26.261us  15.269ms  cudaMemPrefetchAsync
                    0.11%  460.88us         1  460.88us  460.88us  460.88us  cuDeviceTotalMem
                    0.08%  315.33us        96  3.2840us     838ns  101.83us  cuDeviceGetAttribute
                    0.07%  302.34us         1  302.34us  302.34us  302.34us  cuDeviceGetName
                    0.03%  102.04us         1  102.04us  102.04us  102.04us  cudaLaunchKernel
                    0.00%  16.762us         1  16.762us  16.762us  16.762us  cudaGetDevice
                    0.00%  10.406us         1  10.406us  10.406us  10.406us  cuDeviceGetPCIBusId
                    0.00%  4.5380us         3  1.5120us  1.0470us  2.3740us  cuDeviceGetCount
                    0.00%  3.9810us         1  3.9810us  3.9810us  3.9810us  cudaDeviceGetAttribute
                    0.00%  2.5140us         2  1.2570us     978ns  1.5360us  cuDeviceGet
                    0.00%  1.2570us         1  1.2570us  1.2570us  1.2570us  cuDeviceGetUuid
                    0.00%     978ns         1     978ns     978ns     978ns  cudaGetLastError

==36139== Unified Memory profiling result:
Device "GeForce GTX 1060 6GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     192  2.0000MB  2.0000MB  2.0000MB  384.0000MB  34.35110ms  Host To Device
     768  170.67KB  4.0000KB  0.9961MB  128.0000MB  10.70701ms  Device To Host
Total CPU Page faults: 1536
```

Pic. 4. A result of running vector_add_prefetch_gpu with nvprof, which stands for unified memory is being accessed only by the GPU.

Afterward, we wanted to enhance our performance even more. For this purpose, we added a function called initWith, which initializes data on GPU before prefetching it. Thanks to that operation, the Host to Device data migration was no longer needed (pic. 5). This improvement also reduced the number of CPU page faults to 384. As a consequence of these modifications, we downsized the run-time of our program to around 0.63 seconds (two times less than run-time of vector_add_standard).



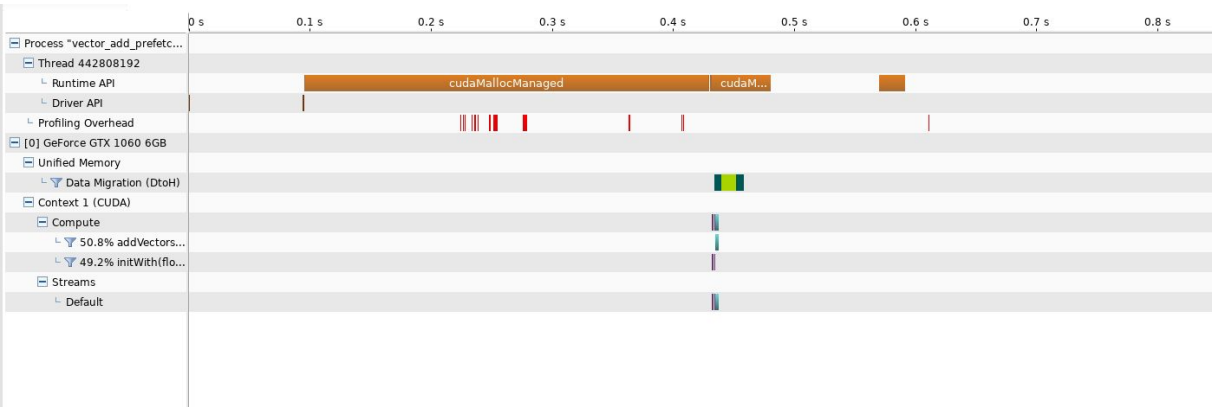
Pic. 5. The window of Nvidia Visual Profiler running vector_add_prefetch_gpu_init_gpu which stands for unified memory is being accessed by the GPU after initializing data on GPU.

```
[cuda-s18@lhcbgpu1 memory_management_lab]$ nvprof ./vector_add_prefetch_gpu_init_gpu
==37681== NVPROF is profiling process 37681, command: ./vector_add_prefetch_gpu_init_gpu
Success! All values calculated correctly.
==37681== Profiling application: ./vector_add_prefetch_gpu_init_gpu
==37681== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   50.74%  2.5681ms         1  2.5681ms  2.5681ms  2.5681ms  addVectorsInto(float*,
float*, float*, int)
                   49.26%  2.4928ms         3  830.94us  829.09us  833.41us  initWith(float, float*,
 int)
      API calls:   91.75%  331.22ms         3  110.41ms  38.203us  331.11ms  cudaMallocManaged
                    5.73%  20.680ms         3  6.8935ms  4.7806ms  11.076ms  cudaFree
                    1.39%  5.0282ms         1  5.0282ms  5.0282ms  5.0282ms  cudaDeviceSynchronize
                    0.85%  3.0690ms         3  1.0230ms  1.0135ms  1.0364ms  cudaMemPrefetchAsync
                    0.13%  460.95us         1  460.95us  460.95us  460.95us  cuDeviceTotalMem
                    0.09%  323.44us        96  3.3690us     838ns  104.83us  cuDeviceGetAttribute
                    0.03%  106.72us         4  26.679us  9.4290us  74.800us  cudaLaunchKernel
                    0.02%  83.949us         1  83.949us  83.949us  83.949us  cuDeviceGetName
                    0.00%  15.714us         1  15.714us  15.714us  15.714us  cudaGetDevice
                    0.00%  9.7080us         1  9.7080us  9.7080us  9.7080us  cuDeviceGetPCIBusId
                    0.00%  4.6780us         3  1.5590us  1.0470us  2.5140us  cuDeviceGetCount
                    0.00%  2.5840us         2  1.2920us  1.0470us  1.5370us  cuDeviceGet
                    0.00%  1.6060us         1  1.6060us  1.6060us  1.6060us  cudaDeviceGetAttribute
                    0.00%  1.2570us         1  1.2570us  1.2570us  1.2570us  cuDeviceGetUuid
                    0.00%  1.0480us         1  1.0480us  1.0480us  1.0480us  cudaGetLastError

==37681== Unified Memory profiling result:
Device "GeForce GTX 1060 6GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     768  170.67KB  4.0000KB  0.9961MB  128.0000MB  10.70598ms  Device To Host
Total CPU Page faults: 384
```

Pic. 6. The result of running vector_add_prefetch_gpu_init_gpu with nvprof, which stands for unified memory is being accessed by the GPU after initializing data on GPU.

As a final modification, we implemented prefetching data to CPU as well as to GPU. We also included in code the function called initWith, which was used before in vector_add_prefetch_gpu_init_gpu. As may be noted at the pic. 7 and pic. 8 these operations reduced the run-time of our program to less than 6 seconds. This happened mostly due to the complete removal of CPU page faults, which previously had caused elongation of the operating time of our program.



Pic. 7. The window of Nvidia Visual Profiler running vector_add_prefetch_gpucpu_init_gpu which stands for unified memory is being accessed by the GPU and the CPU after initializing data on GPU.

```
[cuda-s18@lhcbgpu1 memory_management_lab]$ nvprof ./vector_add_prefetch_gpucpu_init_gpu
==37116== NVPROF is profiling process 37116, command: ./vector_add_prefetch_gpucpu_init_gpu
Success! All values calculated correctly.
==37116== Profiling application: ./vector_add_prefetch_gpucpu_init_gpu
==37116== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   50.82%  2.5691ms         1  2.5691ms  2.5691ms  2.5691ms  addVectorsInto(float*,
float*, float*, int)
                  49.18%  2.4861ms         3  828.70us  824.04us  831.62us  initWith(float, float*,
 int)
      API calls:   80.14%  301.86ms         3  100.62ms  23.956us  301.78ms  cudaMallocManaged
                   12.38%  46.638ms         4  11.660ms  949.77us  43.715ms  cudaMemPrefetchAsync
                    5.88%  22.166ms         3  7.3886ms  4.7662ms  12.494ms  cudaFree
                    1.33%  5.0265ms         1  5.0265ms  5.0265ms  5.0265ms  cudaDeviceSynchronize
                    0.13%  470.94us         1  470.94us  470.94us  470.94us  cuDeviceTotalMem
                    0.08%  309.88us        96  3.2270us     838ns  104.27us  cuDeviceGetAttribute
                    0.03%  102.04us         4  25.509us  8.5210us  72.285us  cudaLaunchKernel
                    0.02%  61.181us         1  61.181us  61.181us  61.181us  cuDeviceGetName
                    0.00%  17.111us         1  17.111us  17.111us  17.111us  cudaGetDevice
                    0.00%  9.7780us         1  9.7780us  9.7780us  9.7780us  cuDeviceGetPCIBusId
                    0.00%  5.1690us         3  1.7230us  1.0480us  3.0730us  cuDeviceGetCount
                    0.00%  2.6530us         2  1.3260us  1.0470us  1.6060us  cuDeviceGet
                    0.00%  1.6760us         1  1.6760us  1.6760us  1.6760us  cudaDeviceGetAttribute
                    0.00%  1.3270us         1  1.3270us  1.3270us  1.3270us  cuDeviceGetUuid
                    0.00%     908ns         1     908ns     908ns     908ns  cudaGetLastError

==37116== Unified Memory profiling result:
Device "GeForce GTX 1060 6GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      64  2.0000MB  2.0000MB  2.0000MB  128.0000MB  10.20477ms  Device To Host
```

Pic. 8. The result of running vector_add_prefetch_gpucpu_init_gpu with nvprof, which stands for unified memory is being accessed by the GPU and the CPU after initializing data on GPU.

SUMMARY

- Unified Memory allows us to allocate data which could be read and written on GPUs as well as on CPUs. As a result of this there is no longer needed to allocate data separately on CPU and GPU.
- By prefetching data to CPU and GPU we can reduce the number of page faults.
- By analyzing outputs of cuda profilers we can significantly improve the performance of our program.