Report 4

Introduction to CUDA and OpenCL

SM, memory prefetching and nvprof.

Dariusz Biela

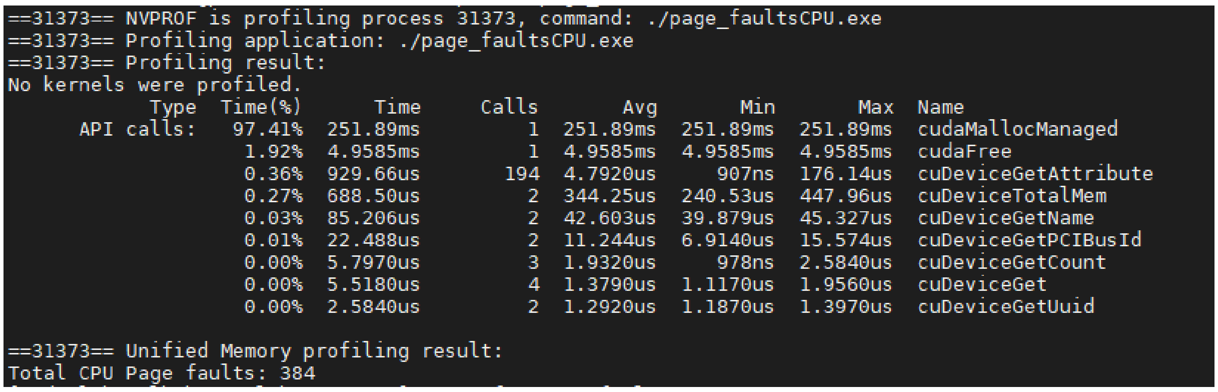
Karol Sawicki

1. Introduction

At the time of our fourth lab classes we were trying to gain more performance with using Unified Memory. To do this we had to learn about memory prefetching mechanism.
Memory prefetching is asynchronous mechanism that sends data from the device to the CPU before it needs the data. This allows to do operations quicker, because when CPU needs the data it already is available. Memory prefetching works in the opposite direction too.
Memory prefetching basically allows hardware to avoid page fault problems. Page fault is a type of hardware exception, that can be occurred when program tries to access memory that is not currently mapped.
As an addition we learned about SM – Streaming Multiprocessors – every nvidia GPU chip is built of them.

2. Calculations and measurements

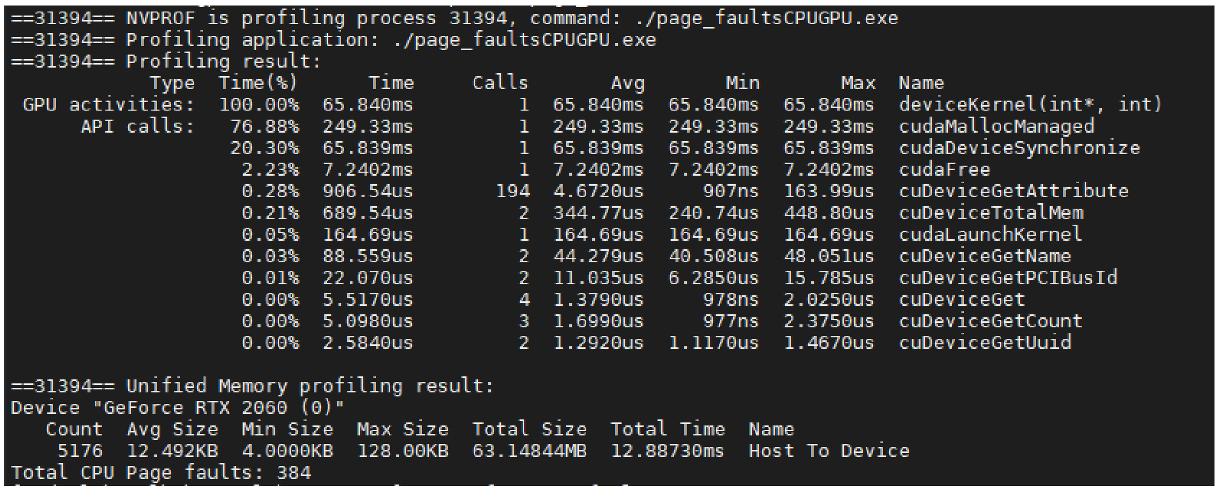
To start investigations we tried to use nvprof tool with the sample codes. We decided to show and discuss our results:



```
==31373== NVPROF is profiling process 31373, command: ./page_faultsCPU.exe
==31373== Profiling application: ./page_faultsCPU.exe
==31373== Profiling result:
No kernels were profiled.
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
      API calls:   97.41%  251.89ms         1  251.89ms  251.89ms  251.89ms  cudaMallocManaged
                    1.92%  4.9585ms         1  4.9585ms  4.9585ms  4.9585ms  cudaFree
                    0.36%  929.66us       194  4.7920us     907ns  176.14us  cuDeviceGetAttribute
                    0.27%  688.50us         2  344.25us  240.53us  447.96us  cuDeviceTotalMem
                    0.03%  85.206us         2  42.603us  39.879us  45.327us  cuDeviceGetName
                    0.01%  22.488us         2  11.244us  6.9140us  15.574us  cuDeviceGetPCIBusId
                    0.00%  5.7970us         3  1.9320us     978ns  2.5840us  cuDeviceGetCount
                    0.00%  5.5180us         4  1.3790us  1.1170us  1.9560us  cuDeviceGet
                    0.00%  2.5840us         2  1.2920us  1.1870us  1.3970us  cuDeviceGetUuid

==31373== Unified Memory profiling result:
Total CPU Page faults: 384
```

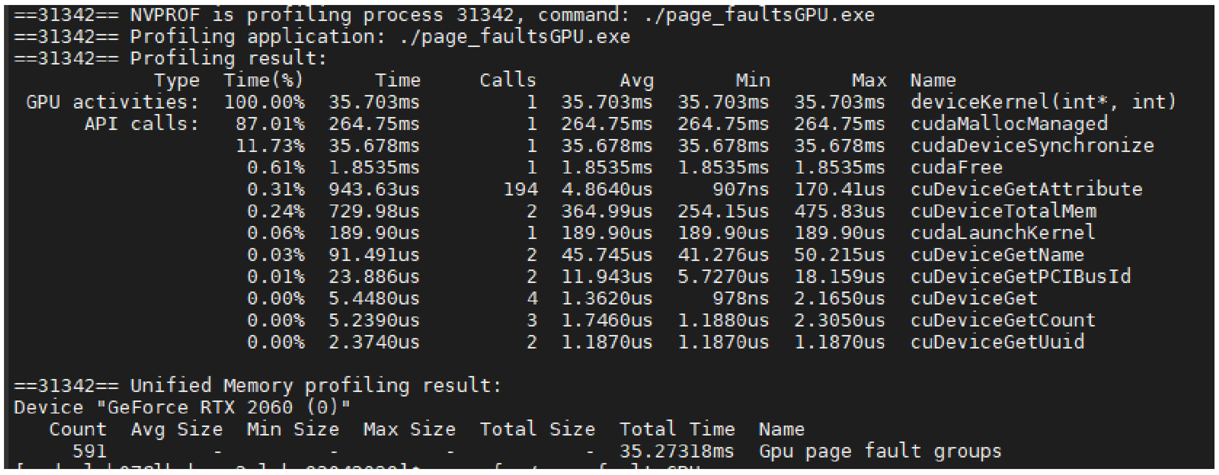
Picture 1 CPU only page faults



```
==31394== NVPROF is profiling process 31394, command: ./page_faultsCPUGPU.exe
==31394== Profiling application: ./page_faultsCPUGPU.exe
==31394== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  65.840ms         1  65.840ms  65.840ms  65.840ms  deviceKernel(int*, int)
      API calls:   76.88%  249.33ms         1  249.33ms  249.33ms  249.33ms  cudaMallocManaged
                   20.30%  65.839ms         1  65.839ms  65.839ms  65.839ms  cudaDeviceSynchronize
                    2.23%  7.2402ms         1  7.2402ms  7.2402ms  7.2402ms  cudaFree
                    0.28%  906.54us       194  4.6720us     907ns  163.99us  cuDeviceGetAttribute
                    0.21%  689.54us         2  344.77us  240.74us  448.80us  cuDeviceTotalMem
                    0.05%  164.69us         1  164.69us  164.69us  164.69us  cudaLaunchKernel
                    0.03%  88.559us         2  44.279us  40.508us  48.051us  cuDeviceGetName
                    0.01%  22.070us         2  11.035us  6.2850us  15.785us  cuDeviceGetPCIBusId
                    0.00%  5.5170us         4  1.3790us     978ns  2.0250us  cuDeviceGet
                    0.00%  5.0980us         3  1.6990us     977ns  2.3750us  cuDeviceGetCount
                    0.00%  2.5840us         2  1.2920us  1.1170us  1.4670us  cuDeviceGetUuid

==31394== Unified Memory profiling result:
Device "GeForce RTX 2060 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
    5176  12.492KB  4.0000KB  128.00KB  63.14844MB  12.88730ms  Host To Device
Total CPU Page faults: 384
```

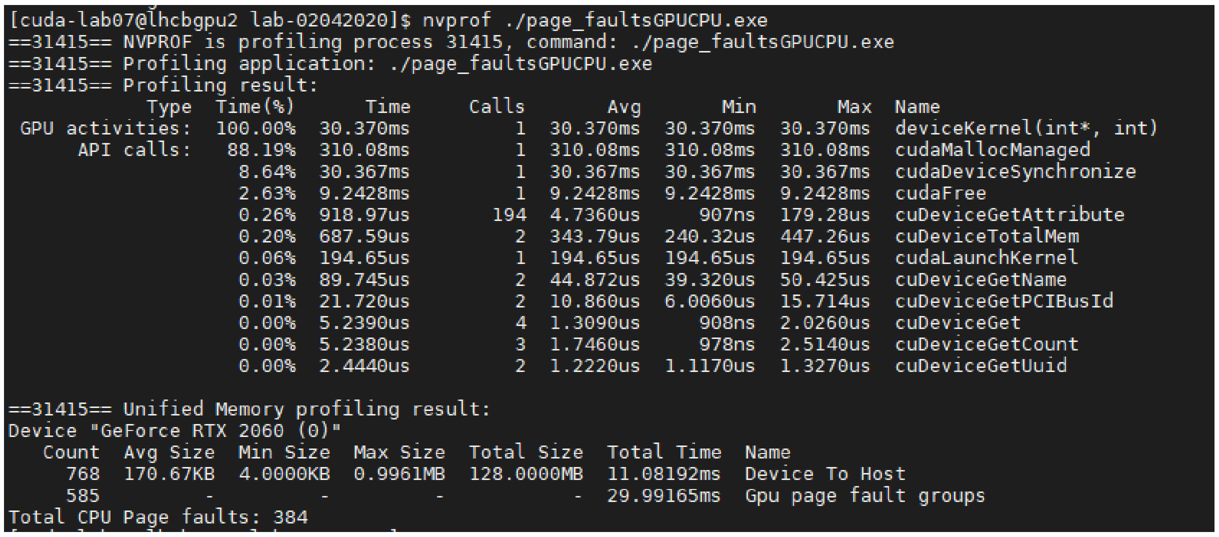
Picture 2 CPU to GPU page faults



```
==31342== NVPROF is profiling process 31342, command: ./page_faultsGPU.exe
==31342== Profiling application: ./page_faultsGPU.exe
==31342== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  35.703ms         1  35.703ms  35.703ms  35.703ms  deviceKernel(int*, int)
      API calls:   87.01%  264.75ms         1  264.75ms  264.75ms  264.75ms  cudaMallocManaged
                   11.73%  35.678ms         1  35.678ms  35.678ms  35.678ms  cudaDeviceSynchronize
                    0.61%  1.8535ms         1  1.8535ms  1.8535ms  1.8535ms  cudaFree
                    0.31%  943.63us       194  4.8640us     907ns  170.41us  cuDeviceGetAttribute
                    0.24%  729.98us         2  364.99us  254.15us  475.83us  cuDeviceTotalMem
                    0.06%  189.90us         1  189.90us  189.90us  189.90us  cudaLaunchKernel
                    0.03%  91.491us         2  45.745us  41.276us  50.215us  cuDeviceGetName
                    0.01%  23.886us         2  11.943us  5.7270us  18.159us  cuDeviceGetPCIBusId
                    0.00%  5.4480us         4  1.3620us     978ns  2.1650us  cuDeviceGet
                    0.00%  5.2390us         3  1.7460us  1.1880us  2.3050us  cuDeviceGetCount
                    0.00%  2.3740us         2  1.1870us  1.1870us  1.1870us  cuDeviceGetUuid

==31342== Unified Memory profiling result:
Device "GeForce RTX 2060 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     591         -         -         -           -  35.27318ms  Gpu page fault groups
```

Picture 3 GPU only page faults



```
[cuda-lab07@lhcbgpu2 lab-02042020]$ nvprof ./page_faultsGPUCPU.exe
==31415== NVPROF is profiling process 31415, command: ./page_faultsGPUCPU.exe
==31415== Profiling application: ./page_faultsGPUCPU.exe
==31415== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  30.370ms         1  30.370ms  30.370ms  30.370ms  deviceKernel(int*, int)
      API calls:   88.19%  310.08ms         1  310.08ms  310.08ms  310.08ms  cudaMallocManaged
                    8.64%  30.367ms         1  30.367ms  30.367ms  30.367ms  cudaDeviceSynchronize
                    2.63%  9.2428ms         1  9.2428ms  9.2428ms  9.2428ms  cudaFree
                    0.26%  918.97us       194  4.7360us     907ns  179.28us  cuDeviceGetAttribute
                    0.20%  687.59us         2  343.79us  240.32us  447.26us  cuDeviceTotalMem
                    0.06%  194.65us         1  194.65us  194.65us  194.65us  cudaLaunchKernel
                    0.03%  89.745us         2  44.872us  39.320us  50.425us  cuDeviceGetName
                    0.01%  21.720us         2  10.860us  6.0060us  15.714us  cuDeviceGetPCIBusId
                    0.00%  5.2390us         4  1.3090us     908ns  2.0260us  cuDeviceGet
                    0.00%  5.2380us         3  1.7460us     978ns  2.5140us  cuDeviceGetCount
                    0.00%  2.4440us         2  1.2220us  1.1170us  1.3270us  cuDeviceGetUuid

==31415== Unified Memory profiling result:
Device "GeForce RTX 2060 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     768  170.67KB  4.0000KB  0.9961MB  128.0000MB  11.08192ms  Device To Host
     585         -         -         -           -  29.99165ms  Gpu page fault groups
Total CPU Page faults: 384
```

Picture 4 GPU to CPU page faults

Pictures from *1* to *4* are describing nvprof results when exemplary code was profiled. We can see that CPU page faults are always the same – when they are. The more interesting are GPU page faults – in each case number of page fault groups is different and every time we see time that we are losing with these page faults. This lost time could be saved by good memory prefetching use.

3. Conclusions

Our nvprof tool shows us that page faults problems are real world problems and – when they are – we could lose so much time. The *Picture 3* shows that we lost *35.3 ms* – that may seem really small amount of lost time, but when we look at the kernel execution time - *35.7 ms* – we know that it is really problematic, especially when we have to execute kernel hundreds of thousands times.

Usage of memory prefetching can be helpful to avoid page faults, especially when we know when data is needed by the host or nvidia device that we are using. In the past, with older GPU chips every operation was ended with memory synchronization even if that was not necessary. Today we can use prefetching with unified memory only when we really have to synchronize our data and avoid page faults, that cause time loss.