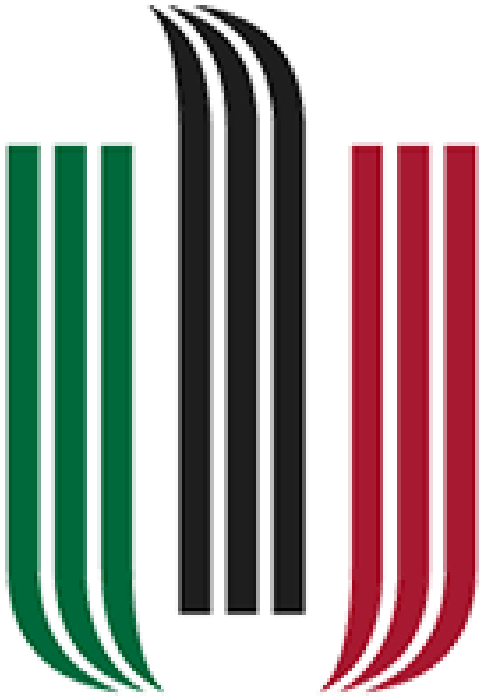


AGH

AGH University of Science and Technology

Introduction to CUDA and Open CL

Lab 3

Michał Kunkel
Wiktor Żychowicz

1. Page fault.

This is an exception when running program trying to access a memory page isn't mapped by the memory management unit. When it happens, kernel tries to reach the required page at the location in physical memory, or crushes the program due to an illegal memory access.

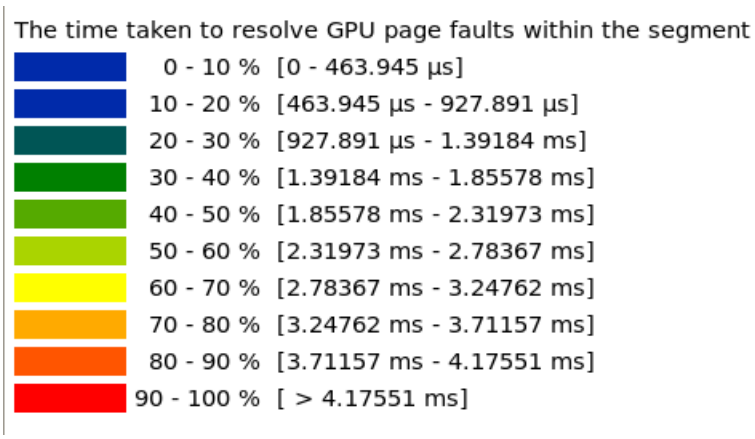
Naming may wrongly suggest that page fault is something bad, but in fact page fault are not always errors. This is the common way for hardware to enlarge memory available to programs in any operating system that uses virtual memory.

On labs we were supposed to research how CPU and GPU behaves when they meet page faults(there are consequence of using cudaMallocMenager and "Unified memory") by modifying this code:

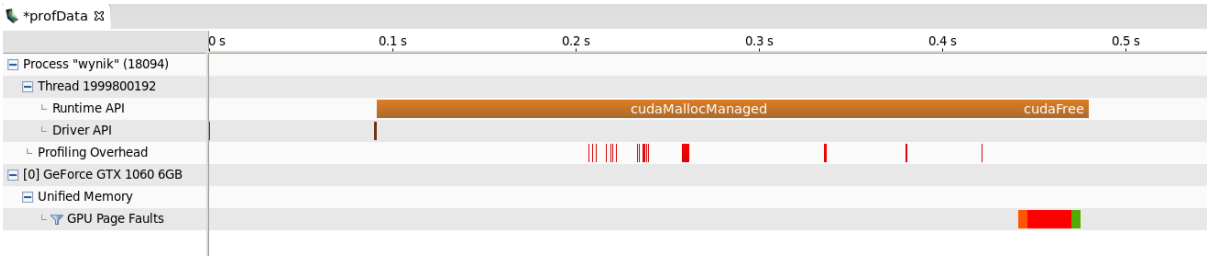
```
__global__
void deviceKernel(int *a, int N)
{
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  int stride = blockDim.x * gridDim.x;

  for (int i = idx; i < N; i += stride)
  {
    a[i] = 1;
  }
}

void hostFunction(int *a, int N)
{
  for (int i = 0; i < N; ++i)
  {
    a[i] = 1;
  }
}

int main()
{

  int N = 2<<24;
  size_t size = N * sizeof(int);
  int *a;
  cudaMallocManaged(&a, size);

  /*
   * Conduct experiments to learn more about the behavior of
   * `cudaMallocManaged`.
   *
   * 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?
   *
   * Hypothesize about UM behavior, page faulting specificially, before each
   * experiement, and then verify by running `nvprof`.
   */

  cudaFree(a);
}
~
~
~
"memory_page_faults.cu" 44L, 953C
```

Picture 1: Source code of a program on which we tested.

2. Examples.

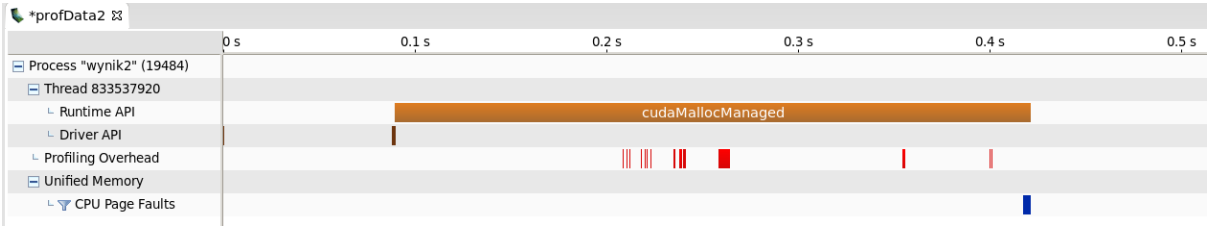


Picture 2.0: Legend of colours in NVIDIA profiler.



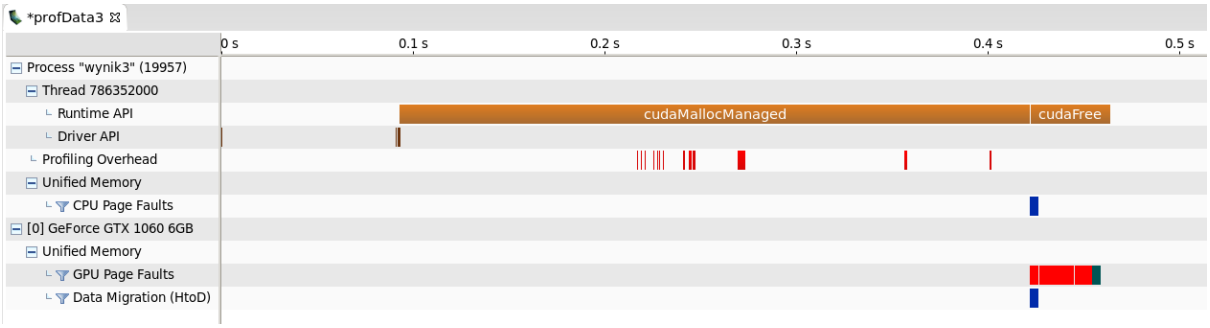
Picture 2.1: Unified memory accessed only by the GPU.

In first case kernel takes a lot of time to overcome (over 4 ms) page fault problems. Despite of them program compiles and works as it should. They do harm only to performance.



Picture 2.2: Unified memory accessed only by the CPU.

In second case kernel managed to reached to problematic area of memory and managed to deal with page fault and for CPU it took less then 1ms. When we compare this example to others we can clearly see that CPU deals with page faults faster than GPU.



Picture 2.3: Unified memory accessed first by GPU then the CPU

In third case we can observe that CPU can help GPU in reaching inaccessible location in physical memory for GPU. After 0.01-0.05 seconds. As we can see to resolve this page fault some data movement is required.

No picture 2.4: Unified memory accessed first by CPU then the GPU

In that case when we execute file we get a segmentation fault which causes it to crash. Sometimes page faults can be fatal to our programs.

3. Some optimization.

Finally we play with some ways we can optimize code. We took simple program which sums up vectors. In all cases we use ‘stride“. It Is a strategy of doing loops in parallel manner where we instead of launching one thread to one iteration we use in-thread loop to take next elements when it end executing previous(good size of step is gridDim * blockDim) without waiting for other threads. We tested asynchronized prefetching of data and result. We also check how place where we initialize vectors affect efficiency.