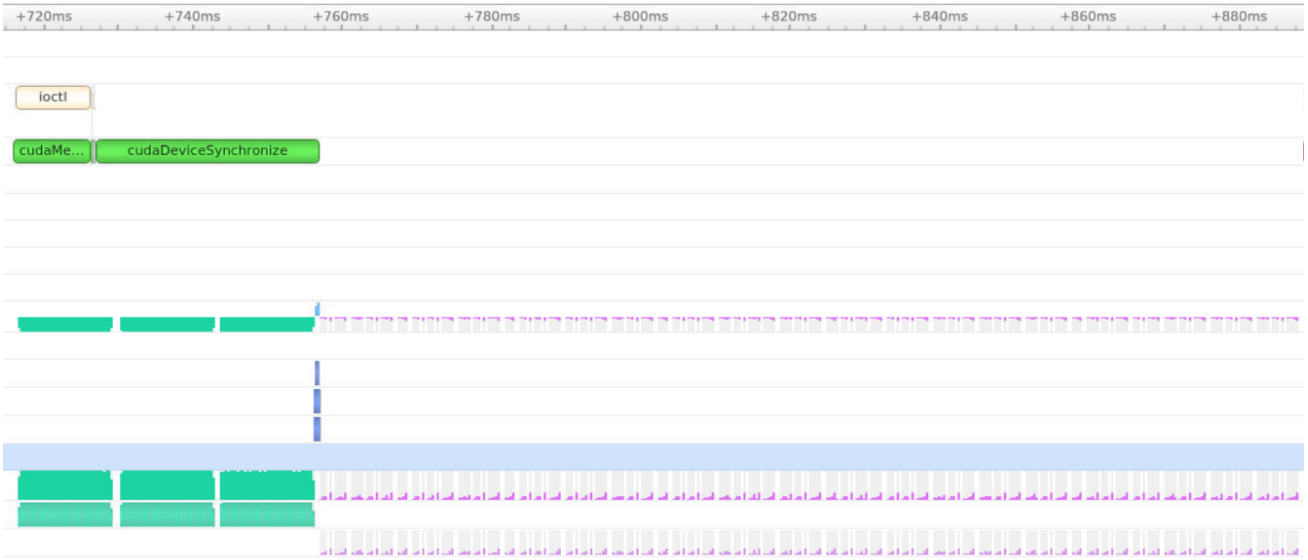
CUDA Memory Management and Code Profiling with NVIDIA Nsys GUI

In this report I will take a closer look at how important is smart memory management in accelerated programs. The results will be analysed using the NVIDIA Nsys GUI. The program allocates memory for three vectors and adds them using a GPU kernel. I will analyse four different versions of program, each of them with different approach to memory management.

The first version of a program includes no smart memory management. Memory allocated using *cudaMallocManaged* is first migrated to the CPU (page fault occurs) where the values are initialized. Then when the kernel is launched, on-demand memory migration takes place and the memory is sent to the GPU. After kernel finished its execution, the memory is migrated back to the CPU.

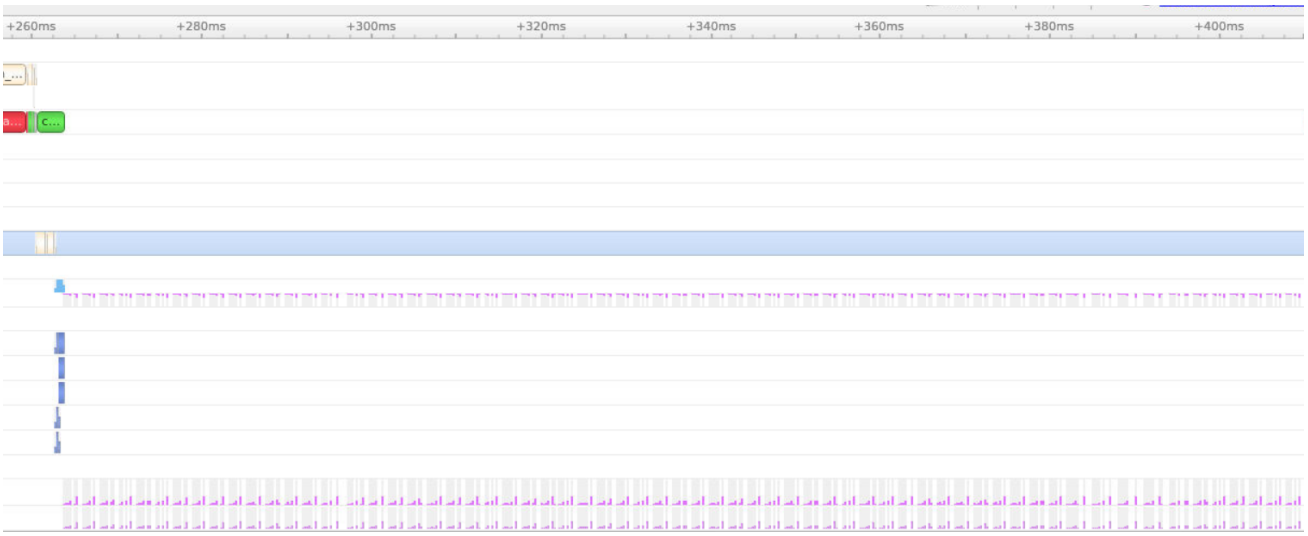
In the picture above we can see how long memory migration from Host to Device takes place (cyan charts, ~721ms - ~821ms) and back from the Device to the Host (purple charts, ~823ms - ~954ms). It is also noticable that the memory is migrated in small blocks which means there are many operations being performed to transfer it. We can also see that memory migration took almost twice as much time as executing the kernel, which could cause significant loss of time in more complicated programs. The entire program run for around 997ms.

The second version of the program utilizes asynchronous memory prefetching, using *cudaMemPrefetchAsynch().* It is used when memory is supposed to be transfered to the GPU (when Kernel is launched). The advantages of that approach are that the memory is migrated in larger blocks and the process is therefore faster and that everything happens asynchronously.



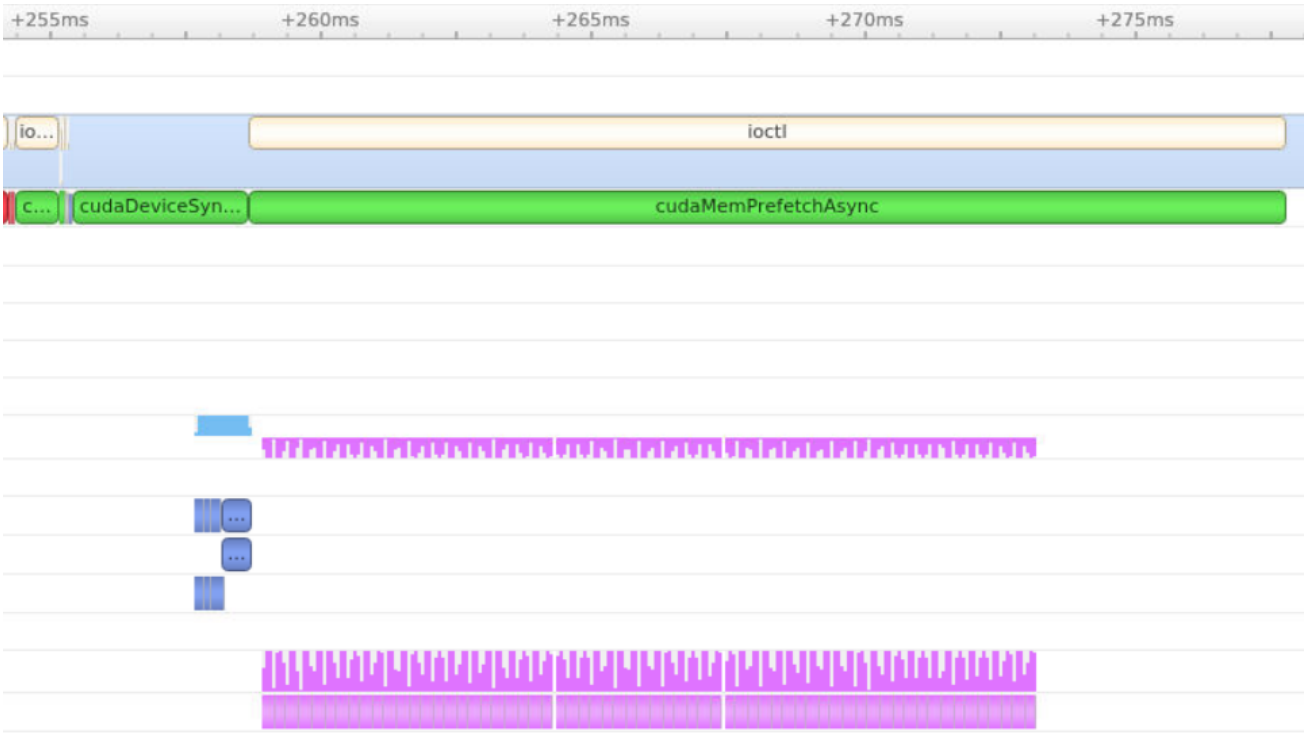
As we can see the amount of time needed to migrate memory using asynchronous prefetching is around 30ms (~716ms - ~756ms). Which is around 3 times faster than in the previous version. The number of operations performed is also decreased. However, the program still suffers from long Device to Host memory migration, which will be improved in the fourth version of this program. The program executes for around 929ms.

In the third approach, we use another kernel to initialize the values of vectors (before that happens we prefetch the memory to the GPU). It is a great improvement ot the program because the memory can stay on the Device for the duration of executing kernels and after that's done it is transfered back to the Host using on-demand memory migration.



As we can see the program is already much better optimized (it is 2 times faster than previous versions, 441ms) which shows how important is smart memory management in GPU accelerated programs. There is still one more way to improve it, because the on-demand memory migration from Device to Host still is an issue (~260ms - ~308ms).

The fourth and final program, will show the true potential of smart memory management by first prefetching the memory to the Device where it is initialized and calculated and then prefetching it back to the Host.



The program runs for around 390ms which is 2.5 times faster than the first version and was achieved without optimizing the kernels or the execution configuration but migrating memory asynchronously and without default operations. The picture above shows that transfering memory from Device to Host using asynchronous prefetching is much more efficient and takes only around 20ms (larger memory blocks = less operations = efficiency).

All of the above approaches show how only a few modifications to a program can increase its performance and how important are memory operations when the program uses several devices.