Steams_Lab

December 6, 2023

Massive parallel programming on GPUs and applications, by Lokman ABBAS TURKI

1 15. Concurrency and asynchronous data transfer

1.1 15.1 Objective

In large applications, involving regular data transfer and calling multiple independent kernels at the same time on the GPU, the concurrency of kernel execution and making the data available in the right memory at the right time become crucial. Indeed, GPUs started to be very efficient SIMD (Single Instruction Multiple Data) processors and they became MIMD (Multiple Instruction Multiple Data) processors with large computing resources and bandwidth. It becomes then complicated to feed them with few kernels/data and keep them busy. Using streams on GPUs offers a good solution for managing data transfer and concurrency. The main purpose of this lab is to start using streams with a simple example.

Students are encouraged to use the CUDA documentation, in particular:

1) the specifications of CUDA API functions within the CUDA_Runtime_API.
2) the examples of how to use the CUDA API functions in CUDA_C_Programming_Guide

1.2 15.2 Content

Compile StreamTest.cu using

[ ]:
```
!nvcc StreamTest.cu -o ST
```

Execute ST using (on Windows machine ./ is not needed)

[ ]:
```
!./ST
```

As long as you did not include any additional instruction in the file Add.cu, the execution above is supposed to return

a[i]+b[i] = 399507447, 399507447
a[i]+b[i] = 399507450, 399507450
a[i]+b[i] = 399507453, 399507453
Processing time for doing everything once: 137.347748 ms
a[i]+b[i] = 399507447, 399507447
a[i]+b[i] = 399507450, 399507450
a[i]+b[i] = 399507453, 399507453
Processing time for doing everything once: 136.145309 ms

a[i]+b[i] = 399507447, 399507447
a[i]+b[i] = 399507450, 399507450
a[i]+b[i] = 399507453, 399507453

Of course, the execution time changes at each call and depends on the machine's performance.

When launched with -1, the function withoutStream performs standard memory copy and execution of the kernel. When launched with a positive integer, the function withoutStream splits the overall copy and kernel call into smaller frames copy and multiple kernel calls.

1.2.1 15.2.1 Using streams, withStream function

a) Using cudaDeviceGetAttribute, check if the device can concurrently copy memory while executing a kernel and if the device can concurrently execute multiple kernels simultaneously,

b) With cudaStream_t define your streams and create them with cudaStreamCreate. At the end of function withStream, use cudaStreamDestroy to destroy streams.

c) Use «<(F+1023)/1024,1024,0,stream[j]»> to call kernels and cudaMemcpyAsync to copy data

d) Do you see any benefits from using streams?

1.2.2 15.2.2 The impact of the size of the data and the length of the kernel

In all these questions, use !nvprof ./ST

a) Execute ST for the different sizes provided in the main function

b) Uncomment the for loop in add_k and repeat a).

c) Replace 50 with 1000 in the for loop in add_k and repeat a).

[ ]: