Learning how to handle errors that occur in a CUDA program:

Graphical user interface, text, application

Description automatically generatedThere are two types of errors in regular code:

However, in CUDA we work with 2 physically separated hardware. We need to transfer errors **that happened in the device** to the ***host,***

Thus, almost all CUDA functions (except for kernel) returns cudaError. The previous is an ***enum*** that is defined in the CUDA API.

If the return value is cudaSuccess, that means the operation we perform in the device is successful. Otherwise, it will return anything but the previous.

Graphical user interface, text, application

Description automatically generatedThen, we pass the returned cudaError to cudaGetErrorString function in this way to get the text corresponding to the error code.

In this course, we skip this error checks to make the example plainer and cleaner. However, in this industry the error check must be used (potential fire reason).

Learning a simple way to measure execution time:

We will note which **CPU clock cycle** we are at before and after the operation. Then, the *difference the two will give us the elapsed clock cycles between operations*. Finally, we **divide the difference by clock cycles per second** to **get the number of seconds elapsed during the operation**. Just keep in mind that the values happen really fast, so we need to Graphical user interface, text

Description automatically generatedget the right value for the following formula:

Graphical user interface, text, application

Description automatically generatedDepending on what we want to accomplish, we will have the following performance tests:

Text

Description automatically generatedIn CUDA, it is a common practice to run same implementation multiple times with different block configurations and use the best configuration based on execution time of those runs. We use trial and error to do this (similar on how weight and bias are discovered.)

With:

128 🡪0.059000 - 0.061000

256 🡪0.058000 - 0. 059000

512 🡪0.058000 - 0.062000

1024 🡪0.060000 or 0.063000

Learning the different CUDA Device properties

NVIDIA released the first CUDA (CUDA Toolkit 1.0) version back in 2007.

We have different **set of CUDA capable devices** which we referred to as ***CUDA Computer Capability*** that ranges from ***1 to 7.***Depending on the CUDA device compute capabilities, ***the basic properties of a CUDA Capable Device*** will ***vary***.

Example:

The maximum # of blocks in a grid differs from a CUDA Device with Compute Capability 1 and one with Compute Capability 5.

Also, shared memory size which you can use from each thread block and allowed caches configuration properties ***varies as well***.

When we need to execute CUDA program on the device with different compute capabilities, we need **a way to query a device dynamically** and set above mention properties based on ***device Compute Capability*** ***on the flight***.

**Table

Description automatically generatedTo give you a heads up let me summarize some of the important set of properties here:**

***Name*** 🡪an ASCII string, **identifying the device**.

***Major and minor*** 🡪 the major and minor revision numbers ***defining the device***

***compute capability***. When you check the compute capability for our device, we saw values like **3.2, 5.2, 5.3** values. **Major** is the whole one and **minor** is the fraction point in those values.

***Ex***: **Compute Capability** = **5.2** | ***Major Revision Number*** = **5** & ***Minor Revision Number*** = **2**

***TotalGlobalMem🡪*** the **total amount of global memory available on device in bytes**.

***Max threads per block*** 🡪 the **maximum number of threads per block**. Usually for modern devices this will be **1024**. Notice this value is ***not for single dimension***, but the ***summation of all threads arranges in all three dimensions should be less than this value***.

***maxThreadsDim 🡪*** a dim3 type property and it indicate **maximum size of each**

**dimension of a block**. But remember, ***summation of all the threads should adhere the restriction we defined in the previous property.***

***MaxGridSize 🡪***  a dim3 type property and it indicates **maximum size of each dimension of a grid**.

***Clock rate 🡪***  gives you **the clock frequency in kilo hertz.**

***Shared mem per block 🡪*** gives you **maximum amount of shared memory available to thread blocks in bytes.**

***Warp Size 🡪* the basic units of execution in streaming multiprocessor** and ***for all the devices warp size is currently 32***.