GPUs: Practical session 1

1 Notes on the practicals

This series of practicals is designed to help you understand how to use NVIDIA's CUDA language and the various tools that are available to help in software development. More detailed examples than are given in these practicals can be found in the SDK examples. Some of these are also accompanied by PDFs giving detailed comments on how they work.

For best results, you should do these practicals on a desktop or laptop equipped with a CUDA-enabled graphics card. However, since CUDA-enabled graphics cards for laptops are only available in high-end machines, you will be able to login to a remote GPU server so that you can run the examples.

2 Getting the code

You can download the practicals from Moodle and unzip them:

```
unzip CUDA_Practicals.zip
```

Note that there are also sample solutions (in `Practicals/solutions`). However, you should resist the temptation to look at these! In any case, you may come up with solutions different to those provided.

3 Suitable machines

Not all of the machines on the CSC network have CUDA-enabled GPUs. Those that have a reasonable capability ($\geqslant 3.5$) are:
`tycho.lsc.phy.private.cam.ac.uk`
Most of the desktops accessible to CSC students have a card with Compute Capability 5.0.

4 Installing CUDA and supporting material

This section covers setting up CUDA on your own (non-CDT) laptop (only for those with CUDA-enabled systems), and making sure you can run on our GPU server.

See `https://developer.nvidia.com/cuda-downloads` for details on how to install CUDA for your own laptop. You should first check whether you have an NVIDIA GPU or not. For some systems, there may be particular issues.

4.1 Ubuntu

Ubuntu has the NVIDIA drivers available as separate packages: `nvidia-common`, `nvidia-settings`, `nvidia-compute-utils-495`, `nvidia-driver-495` although you should check for the latest driver version that works with your hardware at `https://developer.nvidia.com/cuda-downloads`.

Once these are installed, you can use the `cuda_11.5.0_495.29.05_linux.run` download from NVIDIA and install only the toolkit and the SDK (use the `--help` option).

Laptops with on-board Intel GPUs may need particular care; support has improved in recent years, but performance may suffer if using the NVIDIA GPU.

5 Connecting to the remote GPU server

In order to connect to the remote GPU server, open a terminal and type:

```
ssh username@tycho.lsc.phy.private.cam.ac.uk
```

making sure that you are logged in to UniOfCam/Eduroam as appropriate. You could also `ssh` into another CSC desktop, as listed at:
`https://www-internal.lsc.phy.cam.ac.uk/systems`

6 Running CUDA on your laptop

If you have a CDT laptop, then the appropriate libraries and drivers are pre-installed. When referring to paths below, use `/opt/cuda-11.2` instead of `/lsc/opt/cuda-11.2`.

6.1 Server information

If on `tycho` you have access to a machine with two Tesla cards (capability 3.5), and CUDA Toolkit 11.2 is installed.

Although there will be several of you using the server at once, and this could affect the speed of your codes, only one kernel can in fact be run on the GPU at once.

7 Compiler and documentation

The latest compiler is installed under `/lsc/opt/cuda-11.2/bin` and the documentation under `/lsc/opt/cuda-11.2/doc/pdf`. It is worth looking at the `CUDA_C_Programming_Guide.pdf` and `CUDA_Runtime_API.pdf`, as well as some of the other documents.

8 Trying the NVIDIA SDK

As a starting point, the NVIDIA SDK has been installed and compiled into `/lsc/opt/cuda-11.2-samples/`.

8.1 Check device parameters

Run `deviceQuery` (from a directory into which you can write):

```
/lsc/opt/cuda-11.2-samples/1_Utilities/deviceQuery/deviceQuery
```

Make a note of the various device characteristics, and check that they match with that given in the CUDA documentation. Next, run `1_Utilities/bandwidthTest/bandwidthTest` and make a note of the output. You can use these figures to compare to performance metrics later in the practical.

9 Environment Check

You should make sure that the appropriate compiler and libraries are available in your `$PATH`, for example:

```
export PATH=/lsc/opt/cuda-11.2/bin:$PATH
export LD_LIBRARY_PATH=/lsc/opt/cuda-11.2/lib64:$LD_LIBRARY_PATH
```

Check that you are running version 11.2 of the compiler by doing:

```
nvcc --version
```

10 Compile a CUDA code

We'll now try compiling a simple CUDA code. Download and unpack the practicals tar-ball, go into the `Practicals` directory, and type:

```
nvcc sumVectors.cu -o sumVectors -arch=compute_35 -code=sm_35,sm_50,sm_52,sm_53,sm_60,sm_61
```

or use:

```
make sumVectors
```

which will compile using the same flags.

Now run `./sumVectors`. You will be asked for a vector size. Try a variety of values, from 10 to 10^9. The code then outputs the bandwidth, which is the rate of transfer of data from GPU memory to the streaming multiprocessors. It does not take into account the time taken for the actual floating-point sum and all other data transfers.

Examine the `sumVectors.cu` code. It contains most of the elements you need in any basic CUDA code for communicating between the CPU and the GPU.

From now on, all practical examples can be compiled using an instruction such as `make sumVectors`. The Makefile is not particularly complicated; it just saves typing on your part.

To consider:

- Why is the bandwidth low for small vectors?
- What is the largest vector-size (approximately) that you can use?

11 More examples

Look at some more of the SDK examples. These often make use of the `cutil_inline.h` header, containg various macros and functions that are useful in CUDA. I suggest that you do not use these in your own code as they are not part of CUDA itself (and hence not necessarily available if you compile on a different machine) and are subject to change without notice.

In particular, you could look at the following programs, being the simpler ones

- `0_Simple/vectorAdd/vectorAdd`
- `0_Simple/vectorAddDrv/vectorAddDrv` (uses the Driver API - compare to vectorAdd)
- `0_Simple/matrixMul/matrixMul` (see `matrixMul_kernel.cu`)
- `0_Simple/simpleTexture/simpleTexture` (see `simpleTexture_kernel.cu`)

Many of the rest of the examples are more involved, either in terms of the algorithms used, or in terms of the amount of overhead that is employed to make the examples completely robust and general.

Useful SDK examples for determining information about your GPU input:

- `1_Utilities/bandwidthTest/bandWidthTest`
- `1_Utilities/deviceQuery/deviceQuery`
- `1_Utilities/deviceQueryDrv/deviceQueryDrv` (uses the Driver API)

If you have CUDA installed on your own computer or laptop, you may like to try running the following examples as they have pretty graphics.

- `5_Simulations/fluidsGL/fluidsGL`
- `5_Simulations/particles/particles`
- `5_Simulations/smokeParticles/smokeParticles`

It is not possible to run these last tests over an ssh connection as they rely on the GPU being connected directly to the display.