CPDS Laboratory Hands-On
Lab 3: Brief tutorial on CUDA programming model

J.R Herrero

Fall 2022-23

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Notes:

- All files necessary to follow this Hands-on are available in a compressed tar file available from the following location: `/scratch/nas/1/cpds0/sessions/HO-CUDA.tar.gz`. Copy it to your home directory in `boada.ac.upc.edu` and uncompress it with this command line: `"tar -zxvf HO-CUDA.tar.gz"`.
- In addition to the source files, each directory contains files `Makefile` and `job.sh`. You can use `make` in order to compile the codes in the interactive node `boada-[678]`. However, note that the nodes used interactively do not have GPUs attached to them. Therefore, you will need launch the job to the `cuda` queue as `sbatch job.sh` which is already prepared to use the GPUs in `boada-10`. After the job is executed some files will be left in the current directory. Inspect them together with the source codes in order to learn about how things get done in CUDA.
- This Hands-On is limited to a few exemples. There are lots of examples within directory /Soft/cuda/11.2.1/samples at boada. You can copy them to your home directory and compile them there. Note that either you adapt the Makefiles in view of the ones in this hands-on, or need to copy the part of the subdirectoy structure necessary to have `make` find the files which need to be included.

Examples 1

Introduction to CUDA

This hands-on contains a set of examples that has been prepared with the purpose of introducing some of the main constructs in the CUDA programming model for general purpose programming on Nvidia Graphics Processing Units (GPUs).

1.1 deviceQuery

This code allows us to retrieve information about the GPUs available in the system. Other than knowing the number of cores and their grouping in *multiprocessors*, some of the values shown are useful for helping us define the *grid* and the *block of threads*, the amount of *shared memory* or *registers* available.

1.2 Helloworld

There are two codes `hello1.cu` and `hello2.cu`. The former exemplifies how to launch kernels on different devices and synchronize the end of their execution. The latter, shows how each thread selects the work it has to do (on which data elements) using predefined variables which specify the dimensions of the block, the block index and the thread index within the block:

```
int idx = threadIdx.x+blockIdx.x*blockDim.x;
```

and handling of data storage (`cudaMalloc` and `cudaFree`) plus data transfers using `cudaMemcpy` from the host (CPU) to the device (GPU), using parameter `cudaMemcpyHostToDevice`, and viceversa, specifying parameter `cudaMemcpyDeviceToHost`.

1.3 Stencil

This code requires you to complete kernel `stencil2` which is an optimization of `stencil1` by using *shared memory.* As an exercise, you have to complete four statements which have been left with .... You can find the solution in file `Stencil.cu.soln`.
This code also exemplifies how one can measure the execution time of parts of the code by using *events.* While it's true that the CPU code is not optimized, note that the GPU codes are not that complicated and run one order of magnitude faster. Is the usage of shared memory helping? Yes!

1.4 Add: VectorAdd and MatrixAdd

These codes add two vectors or two matrices respectively. Notice how the definition of the grid and block of threads mimics the 1D or 2D data structure on which the threads have to work on, easing the calculation of the array indices. Also, note that the time to allocate and transfer data to the GPU is adding a large overhead compared to the execution time of the kernels.

Examples 2

More advanced examples

As an exemple of more advances work performed in the GPU we will show how reductions can be done.

2.1 Reduction

Notice that blocks of threads cannot synchronize. Thus, sometimes the reduction kernel is called multiple times, cascading the reduction. In other codes available through Internet you'll find them in a loop.

The `Makefile` contains rules for compiling and linking several kernels. However, submission of other kernels for batched execution will require you to edit file `job.sh` and change the version of the kernel in line 14: "KERNEL=01" to the desired number.

Observe the speedup obtained by the different kernels.