Lab 3 - Advanced CUDA

Advanced Architectures

University of Minho

The Lab 3 focus on the development of efficient CUDA code by covering the programming principles that have a relevant impact on performance. Use a cluster node with a NVidia GPU (by specifying the keyword `tesla` in the node characteristics when submitting a job) and do not submit interactive jobs (you should use something like `qsub -qmei -lnodes=1:ppn=??:k20,walltime=10:00 ./run.sh`, replace `??` by the number of CPU virtual cores in the target compute server used in **Ext 3.1** or remove it for the remaining exercises). Compile the code in the frontend.

This lab tutorial includes two homework assignment (Ext 3.1) and two exercises to be solved during the lab class (Lab 3.x). Copy the source files to your home directory in the SeARCH cluster:

```
cp -r /share/cpd/lab3 .
```

Several algorithms will be implemented for both multicore CPUs and CUDA GPUs, to assess the performance benefits of the GPUs, with different problem sizes. When measuring execution times select the best of three measurements and **plot the results on the previous lab spreadsheet**.

To load the compiler in the environment use the one of the following commands (do not forget to also use them in `run.sh`):

GNU Compiler: `module load gcc/5.3.0`.

CUDA Environment: `module load cuda/10.1`.

You can use other GNU/CUDA combinations at your own risk.

3.1 Shared Memory

Goals: to develop skills in accessing shared data and thread synchronisation.

Lab 3.1 Complement this skeleton with the GPU code (kernel) to perform the same task. Remember that the CUDA kernel is coded in such a way that it should be executed by a single thread, using its *id*. The CUDA runtime will assign the kernel to a set of threads when it is called, so no explicit parallelism must be present in the code. Run the GPU code and measure and record the best execution time. Compare the CPU-GPU performance. Listen (**or**

Algorithm 1 Pseudocode for a 1D stencil.

```
for all E in Vector do
    for all Xi in radius R of E do
        OutVector[E] += Xi;
    end for
end for
```

ask) to the suggestions related to the use of the GPU shared memory to improve performance. Implement the suggested optimisations and measure the performance gains.

Ext 3.1 Consider the one dimensional stencil algorithm that operates on vectors. Using the C++/CUDA skeleton provided in the attached file develop an efficient OpenMP implementation of the `stencilCPU` function to run on a Xeon device. Measure and record the best execution time on a spreadsheet for 3, 4 and 8 threads. Plot the speedup of the CUDA vs multicore execution times for a dataset size of 100 MiBytes. Send the plot with a brief description of the results, detailing how the vector size (the `SIZE` define) was calculated, to ampereira@gmail.com.

3.2 Efficient Access to Data

Goals: to comprehend the coalesced memory access concept and develop skills on data reuse.

Lab 3.2 Consider the one dimensional stencil code from the previous exercise. Create two kernels based on the current implementation, with different ways to access elements on the input vector: specifying **(i)** an offset (i.e., with an offset of 5, thread t_0 will start on the fifth element of the vector, t_1 the sixth, and so on), or **(ii)** a stride (i.e., with a stride of 2, thread t_0 will access the vector at position 0, 2, 4, ..., t_1 will access 1, 3, 5, and so on). Handle the memory transfers independently for each different kernel.

Execute the kernels multiple times with up to 4 different offsets and strides and assess their execution times. Plot this data in a spreadsheet to help visualise any patterns. How do you explain the results? How should you improve the code?

Ext 3.2 Send the plot with a brief explanation of behaviour of the kernels to ampereira90@gmail.com.