DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

ADVANCED COMPUTER ARCHITECTURES

Project 3

Abstract: The third lab consists in acceleration of a simple program, by relying on an heterogeneous system composed of a CPU and a GPU. For this purpose, students will exploit the CUDA application programing interface and data-level parallelism to offload computational intensive (regular) tasks to a GPU.

1 INTRODUCTION

For the development of this work, students should use CUDA, a parallel computing platform and application programming interface model created by Nvidia to enable general-purpose computing on their own GPUs. For the purpose of this work, students will use a heterogeneous machine composed of:

- CPU: Intel Core i7-7700K CPU @ 4.20GHz (4 cores, supporting up to 8 concurrent threads)
- GPU: NVIDIA GeForce GTX 1070 (15 SMs, each with 128 CUDA cores – a total of 1920 CUDA cores)
- Memory: Host 16 GB / device 8GB.

To connect to the machine, students can use the following credentials:

Machine name: cuda1.scdeec.tecnico.ulisboa.pt
Username: aac2018
Password: aacgtx1080
Connection protocol: SSH
Folder with the original source code: /home/aac2018/src/

To help the development of the work, students are advised to use the information on the lecture slides, but can also use other information on the web, e.g.,

- https://www.tutorialspoint.com/cuda/cuda_introduction.htm
- David B. Kirk, Wen-mei W. Hwu. "Programming Massively Parallel Processors: A Hands-on Approach", 3rd Ed(s) 1, 2 or 3. Morgan Kauffman.

NVIDIA also provides multiple examples of CUDA applications, which are normally distributed with the CUDA drivers and API. On the target machine, the CUDA examples are available at:

```
/opt/cuda/samples/
```

All applications have been previously compiled and are available at:

```
/opt/cuda/samples/bin/x86_64/linux/release/
```

One particular usefull application is `deviceQuery`, which allows to query the number of available NVIDIA GPUs, as well as its properties. In this case, by running the command:

```
/opt/cuda/samples/bin/x86_64/linux/release/deviceQuery
```

We get the following output shown in Figure 1.

To develop test programs, students should rely on NVIDIA compiler. For this, students can either: (1) re-write the code using one single file that includes both host and device code; or (2) use multiple files, one for host code, others for device code. In either case, always remember that device code must be placed inside a file with extension .cu. If using a single file (option 1) you can compile the code by using the following line:

```
nvcc -O3 -lm <source code.cu> -o <output binary file>
```

```
/opt/cuda/samples/bin/x86_64/linux/release/deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1070"
  CUDA Driver Version / Runtime Version          9.1 / 9.1
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 8119 MBytes (8513585152 bytes)
  (15) Multiprocessors, (128) CUDA Cores/MP:     1920 CUDA Cores
  GPU Max Clock rate:                            1721 MHz (1.72 GHz)
  Memory Clock rate:                             4004 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536),
3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Version =
9.1, NumDevs = 1
Result = PASS
```

Figure 1 - Output of the example `/opt/cuda/samples/bin/x86_64/linux/release/deviceQuery`.

2 WORK PLAN

The objective of this work is to accelerate the original application, available at the host folder "sequencial_code". Accordingly, students are allowed to apply any transformation to the code that may help minimizing the overall execution time.

To measure the execution time, the `clock_gettime()` function can be used (part of the `<time.h>` library). This allows accurate time measurements with nanosecond precision.

To accomplish this objective use the following steps:

1. Create a new folder under /home/aac2018/ with your group number, e.g.

   ```
   mkdir /home/aac2018/group01
   ```

2. Copy the original code to your directory

   ```
   cp /home/aac2018/src/* /home/aac2018/group01/
   ```

3. Analyse the source code and identify data parallelism opportunities.
4. Compile the original code and measure the execution time. In this lab always use the "-O3" flag to maximize the performance of the CPU (single threaded), e.g.,

```
gcc –O3 –lm <original source files> -o <sequential_executable>
```

5. Identify what are the code sections that mostly contribute for the observed execution time.
6. For each identified section, verify whether it features massively levels of data parallelism. If so, develop a kernel to offload the computation to the GPU.
7. Validate that the code executes correctly.
8. Evaluate the performance of your solution and devise solutions to minimize the execution time even further.

3 PROJECT DUE DATE AND REPORT FORMAT

The project should be completed until June 3, with the code and an up to 6 page report being be submitted online (via Fenix). The report should follow the template for the IEEE Computer society journals:

https://www.ieee.org/publications_standards/publications/authors/author_templates.html

The report should include:

- an abstract (single paragraph, 100-200 words) explaining the objective of the work, as well as summarizing the techniques used and the results obtained;
- an introduction section briefly explaining the difference (in architecture) between CPUs and GPUs, as well as identifying how (and in which cases) GPUs can be used to accelerate the execution of an application;
- a section explaining how (and why) the original source code was modified to improve the performance;
- an experimental results section showing the obtained results (support your text with tables or figures); also, and by varying the input size, show the speed-up between the original (sequential) program and the accelerated program;
- a conclusions section.