Assignment Description

1. "Hello World" in CUDA

The following code shows an example CUDA code. Specifically, it is a version of the well-known "Hello World" example:

Code (file `hello.cu`):

```
#include <stdio.h>

const int N = 16;
const int blocksize = 16;

__global__
void hello(char *a, int *b)
{
   a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
   char a[N] = "Hello \0\0\0\0\0\0";
   int b[N] = {15, 10, 6, 0, -11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

   char *ad;
   int *bd;
   const int csize = N*sizeof(char);
   const int isize = N*sizeof(int);

   printf("%s", a);

   cudaMalloc( (void**)&ad, csize );
   cudaMalloc( (void**)&bd, isize );
   cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
   cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

   dim3 dimBlock( blocksize, 1 );
   dim3 dimGrid( 1, 1 );
   hello<<<dimGrid, dimBlock>>>(ad, bd);
   cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
   cudaFree( ad );

   printf("%s\n", a);
   return EXIT_SUCCESS;
}
```

Questions (About the "Hello World" code in CUDA):

1. Describe how the code works, i.e., how does it get to write "Hello World" from the input vectors and the provided kernel.

2. What memory transfer is produced between the host and the device (GPU)?

2. Execution using the Ocelot emulator

Ocelot is an environment that allows us to execute programs written in CUDA, on GPUs and conventional processors. You can find additional information in the following url:

`http://code.google.com/p/gpuocelot/`

These are the basic instructions you have to follow in order to use Ocelot; for the "Hello World" for a GPU (from the login node):

1. Code (file `hello.cu`):

2. Modify the LD_LIBRARY_PATH accordingly

```
export
LD_LIBRARY_PATH=/export/apps/gcc/4.8.2/lib:/export/apps/gcc/4.8.2/lib64:/e
xport/apps/ocelot/lib/:/export/apps/boost/lib/:$LD_LIBRARY_PATH
```

(please note that the export above is a single line – no line breaks)

3. Compile the CUDA code

```
/export/apps/cuda/5.5/bin/nvcc -cuda hello.cu -I
/export/apps/ocelot/include/ocelot/api/interface/ -arch=sm_20
```

4. The previous step will generate a file called `hello.cu.cpp.ii` which will have to be compiled using g++ in order to obtain an executable binary.

```
/export/apps/gcc/4.8.2/bin/g++ -o hello hello.cu.cpp.ii -I
/export/apps/ocelot/include/ocelot/api/interface/ -L
/export/apps/ocelot/lib/ -locelot
```

5. Execute the program

```
 ./hello
```

You can add the modification of LD_LIBRARY_PATH and PATH in your .bashrc in order to simplify these steps.

The execution, by default, will return the following warnings:

```
==Ocelot== WARNING: Failed to find 'configure.ocelot' in current
directory, loading defaults.
==Ocelot== INFO: You may consider adding one if you need to change the
Ocelot target, or runtime options.
```

You can ignore these warning or create a file within the same directory named "configure.ocelot" with, for example:

```
 {
     "ReRoutes": [],
     "GlobalConfiguration": {}
 }
```

Instructions for physical GPUs (optional, no support will be provided)

You can find information about how to install and use the corresponding SDKs for CUDA and OpenCL (for NVIDIA GPUs or ATI, respectively). You can use the following links for some references: http://developer.nvidia.com/cuda-downloads i http://www.khronos.org/

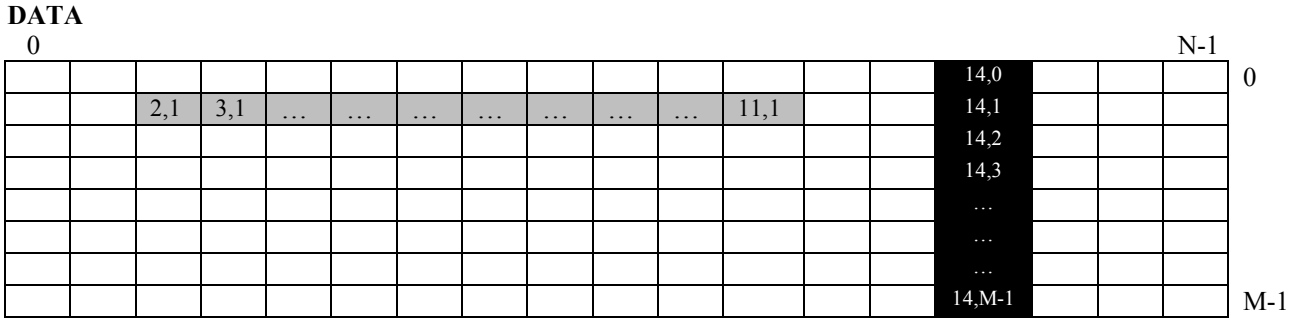
3. Programming exercise

You are requested to implement the following problem using CUDA.

From a **DATA**[N,M] matrix with **N** columns y **M** rows, the program should provide:

1) An output **MOV** matric, where each point MOV[i, j] is the average of DATA[i-9, j], DATA[i-8, j], ..., DATA[i, j] (it represents a "moving average").
2) An **AVG** vector with the mean of the values of all rows for each column, i.e. AVG[i] is the average of DATA[i, 0], DATA[i, 1], ..., DATA[i, M-1].

as shown in the illustration below:



For example:

MOV[11,1] = mean of the grey cell values
AVG[14] = mean of the black cell values

Questions:

3. Provide your implementation using CUDA.

4. Provide an example use of your code. For example, you can generate an input matrix of small size (1000x10) with random values (or following a given distribution) in such a way that you can generate a plot from the input data (plot with 10 series of 1000 points), and the matrix and output vector.

Using CUDA with Data from Cyberinfrastructure

Goals

This part of the assignment aims at exposing students to a cyber-infrastructure (CI) use case by leveraging their CUDA implementation.

It is proposed to use data from the Ocean Observatories Initiative, which delivers marine observation data through its cyber-infrastructure.

OOI data can be obtained using different mechanisms, including the data portal (https://ooinet.oceanobservatories.org) and a machine-to-machine interface (REST API) for programmatic data access.

You can find more information and video tutorials of the OOI data portal in the following link:

https://oceanobservatories.org/data-portal/

From the large number of sensors and data streams available in OOI it is provided sample data from CTD sensors (measure the conductivity, temperature, and pressure of seawater) at different depth points of the water column, and a BOTPT sensor (Bottom Pressure and Tilt Meter, e.g., used to detect earthquakes and predict tsunami and eruptions) at Axial Seamount, a submarine volcano located on the Juan de Fuca Ridge off the coast of Oregon.

A number of data files are provided along with the assignment description which are associated with the following sensors and metrics:

CTD sensor at 194m depth	Water temperature (file 194m_temp.txt)
	Water pressure (file 194m_pres.txt)
CTD sensor at 581m depth	Water temperature (file 581m_temp.txt)
	Water pressure (file 581m_pres.txt)
CTD sensor at 1542m depth	water temperature (file 1542m_temp.txt)
	Water pressure (file 1542m_pres.txt)
CTD sensor at 2903m depth	Water temperature (file 2903m_temp.txt)
	Water pressure (file 2903m_pres.txt)
BOTPT sensor at Axial Seamount	Water pressure (file botpt_pres.txt)

Questions:

4. Use your CUDA implementation with one of the provided datasets (from the previous list such as "water pressure") to illustrate the effect of the "moving average" on the data when they are visualized. You can use different window sizes (e.g., 10, 15 and 20 data points for implementing the "moving average").

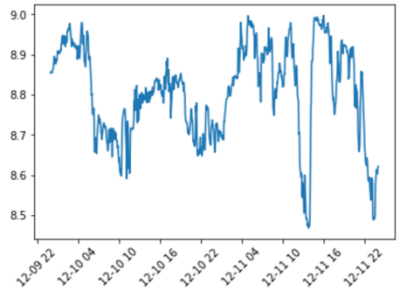
Optional exercises (for improving your score)

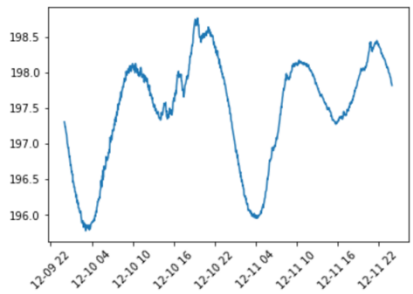
The python script provided (sample.py) is an example for obtaining the sample data. Please note that you will need to replace YOUR_API_USERNAME and YOUR_TOKEN with your user information in the OOI data portal (you can use a google account to log in).

https://vimeo.com/334231662 provides an introduction about the OOI use case. It also discusses Kafka (in case of interest). The password to access the video is "uoc2019"

The python script does also provide simple plotting capabilities as shown below.

```
#CTD 194m

ref_designator = 'CE04OSPS/PC01B/4A-CTDPFA109' #subsite/node/sensor
method = 'streamed'
stream = 'ctdpf_optode_sample'

req = requests.get(prefix_data_query+ref_designator+'/'+method+'/'+stream+'?beginDT='+begin+'&endDT='+end+'&limit='+lim
data = req.json()

df = pd.DataFrame(data)
ntp_diff = (datetime.datetime(1970, 1, 1) - datetime.datetime(1900, 1, 1)).total_seconds()
df['time'] = pd.to_datetime(df.time - ntp_diff, unit='s')
df = df.set_index(df.time)
plt.plot(df['time'], df['seawater_temperature'])
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.show()

plt.plot(df['time'], df['seawater_pressure'])
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.show()

#df['seawater_temperature'].to_csv('194m_temp.txt', index=None,)
#df['seawater_pressure'].to_csv('194m_pres.txt', index=None,)
```





The query requests 1,000 data points between representing the measurements between Dec 10 and Dec 12, 2018 (data is decimated or down-sampled).

You will need to use your environment to run the provided python code. For example, you can use the anaconda framework (multi-platform) - https://anaconda.org

Proposed Optional Exercises

Optional 1. Explore the OOI data portal and OOI website (oceanobservatories.org) to obtain other data sets and analyze them with your CUDA code.

Optional 2. Respond the survey shown below

Please rate (1-5) the following questions in a table as shown below.

1. How much did you know about CI before this course?

2. How much do you know about CI after introducing this topic in this course?

3. Would you like to know more about CI?

4. Do you find useful to use CI in the context of this course?

5. Do you think adding more CI-related topics and activities would improve this course?

6. Would you like to have an assignment/assignments using HPC/CI-related problems (e.g., MPI using python)?

7. Do you have experience using python, data and scientific libraries (e.g., pandas, matplotlib, etc.), GitHub, Jupyter Notebooks, etc.?

8. How positively (5) or negatively (1) do you think including more CI-related topics and related tools (items listed in the previous question) would improve your skills and/or professional competences?

9. How useful do you think including CI-related topics in the textbook materials would be?

10. How useful do you think including formal or informal videos including demos/tutorials and use case scenarios would be in this course?

Question #	1	2	3	4	5	6	7	8	9	10
Rating (1-5)										

Grading Policy

The correct use of the CUDA interfaces/extensions will be evaluated. The use of the performance evaluation tools in your studies as well as your comments will be taken into account.

Submission Format and Due Date

Please submit your assignment as a single PDF containing all the responses including the scripts, graphs etc. Brevity and concretion will be positively considered to grade the assignment.

Submitting a tar or zip file including files with your implementation is allowed (not preferred), but please do NOT use rar.

Assignments should be submitted by **27 May 2020 (hard deadline)**.