Exercise 2: Hardware Accelerators

Task 1. K-means Clustering
This exercise focuses on programming hardware accelerators based on the example of k-means clustering. In general, clustering describes the grouping of a set of objects based on their similarity. The k-means algorithm partitions n data points into k clusters based on the nearest mean of each data point to the clusters. The distance from each data point to the means (often also called centroids) is given by the Euclidean distance $d(p, q)$ of two points $p = (p1, p2)$ and $q = (q1, q2)$ in $\mathbb{R}^2$:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}. \tag{2.1}$$

Each cluster is represented by a single mean vector $m^{(k)}$.
A basic, iterative algorithm is given as follows:

- Initialize the k means randomly.
- Assignment step: each data point is assigned to the nearest mean.
- Update step: each mean is recalculated based on the new assignments.
- Repeat assignment and update steps until algorithm converges.

Task 1.1. Preparations
This exercise can be done on the RWTH cluster environment CLAIX 2018:
Login to one of the frontend nodes of CLAIX (e.g., login18-1.hpc.itc.rwth-aachen.de; see `https://doc.itc.rwth-aachen.de/display/CC` for further information). Task 2 may be done on the GPU Cluster of CLAIX (e.g., login18-g-1.hpc.itc.rwth-aachen.de).
These frontend nodes are meant for login, compiling and short (!) tests. For usual tests and performance measurements, you should always use the batch environment. That means that you submit a "batch job" that contains a description of what you want to do and which is scheduled as soon as there are corresponding resources available. Please note that depending on your requested resources, you will probably have to wait shorter or longer. So, try to request minimal values in runtime, memory and core number. More information can be found in our Primer in chapter 4.4 (see `https://doc.itc.rwth-aachen.de/pages/viewpage.action?pageId=3473705`).
If you have sent your TIM ID to `contact@hpc.rwth-aachen.de`, you have the possibility to submit to a special lecture project (lect0034) that reserves some compute resources for you for the purpose of completing your exercises. Note that these compute resources are for all students in this lecture. Please use them responsibly!
Examples for how to use the batch environment are provided in tasks 2.
Download the sources contained in *ex2_kmeans.tar.gz* from RWTHMoodle and unpack it (*tar xvzf ex2_kmeans.tar.gz*) The archive contains:

- kmeans.cpp: The C++ code base for the whole exercise. The following tasks are marked by todo comments in the code.

- Makefile: Contains the following targets:
  - debug: Build for debugging
  - release: Build for performance
  - run-small: Run kmeans for the small data set
  - run-large: Run kmeans for the large data set
  - vis-small: Generate cluster.gif for the small data set
  - vis-large: Generate cluster.gif for the large data set
  - install-vis: Install requirements for the R visualization script
  - clean: Clean-up of generated files
- input: Contains a small (1000 elements) and a large (1000000 elements) 2D input data set.
- utils: Contains a setup script (*install-prerequisites.sh*) and a visualization script (*vis.R*)

Task 1.2. Implementation of K-means
Implement the proposed k-means algorithm as denoted in the code by *TODO: task 1.2*.

Task 1.3. Deploy: Check for Correctness
Verify that the results obtained in task 1.2 are correct (template: *TODO: task 1.3*).

Task 1.4. Optimization of K-means
If possible, optimize the serial implementation with your knowledge obtained in the lecture without changing the algorithm.
Hint: Think about the used data layouts.

Task 2. K-means Clustering on GPUs
This exercise can be done on the RWTH cluster environment CLAIX 2018:
Login to the frontend node of the GPU Cluster of CLAIX (login18-g-1.hpc.itc.rwth-aachen.de; see `https://doc.itc.rwth-aachen.de/display/CC/GPU+cluster` for further information).
The backend of CLAIX can be used for performance measurements with the following job script:

```
#!/usr/bin/env zsh

### Job name
#SBATCH --job-name=kmeans
#SBATCH --account=lect0034

### File / path where STDOUT will be written, the %j is the job id
#SBATCH --output=output_%j.txt

### Optional: Send mail when job has finished
###SBATCH --mail-type=END
```

```
###SBATCH --mail-user=<email-address>

### Request time and memory
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1500

### Set Queue
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=24
#SBATCH --gres=gpu:volta:1

### Insert the directory where you placed the exercise
cd ~/ex2
./kmeans.exe ./input/large.in 1000000 5000 50
```

This batch script can be submitted to the batch environment by
`$ sbatch kmeans_gpu.sh`
With `squeue -u <TIM>`, you can check whether your job is in the batch queue and whether it is pending or running.
Load the CUDA toolkit to make its tools available:

`module load cuda/100`

To check what GPUs are available to the current system, `nvidia-smi -L` might be used.

Task 2.1. Assess: Investigation of Parallelism
Identify the hotspot of the algorithm of the optimized and checked serial version (task 1.4) and evaluate which parts are suitable for the GPU. Investigate which steps of the algorithm can be parallelized and how it can be achieved. Reason about dependencies.

Task 2.2. Assess: Performance Modeling of K-Means
Model the execution time t_{GPU} of the hotspot on the V100 GPU based on the performance model introduced in the lecture. What limits the execution time of the hotspot?

Task 2.3. Port K-means to GPU
Offload the identified hotspot in task 2.2 to a GPU while taking care of the required data. It is advisable to copy the working serial version to a new file with extension `.cu` (e.g. `cp kmeans.cpp keamns.cu`). This file can then be compiled with:

```
Release:  nvcc -O3 -std=c++11 kmeans.cu -o kmeans_gpu.exe
Debug:  nvcc -g -G -std=c++11 kmeans.cu -o kmeans_gpu.exe
```

Hint: The offloading and optimization steps provided in the lecture might act as a brief guide here. More detailed information can be found in the CUDA documentation platforms provided in the lecture.
Hint 2: Remember to verify your results.

Task 2.4. Optimize and Parallelize K-means on the GPU

Optimize the algorithm and parallelize it to utilize the GPU as much as possible. Keep the data layout optimizations from task 1.4 in mind.

Task 2.5. Deploy: Verify Results
Check you results for correctness and evaluate the actual performance with the performance estimated with your model in task 2.2. How close is you implementation to the theoretical peak performance for that hotspot based on the model? Evaluate reasons for potential differences between the modeled and the achieved performance.

Task 2.6. Performance Comparison
Compare your obtained performance results of the k-means algorithm on the GPU to the ones on the CPU (task 1.4). Is this problem suitable for the GPU (meaning: is the algorithm accelerated by the usage of the GPU)? Justify your answer.
Is this comparison fair? Justify your answer.

Discussion of this exercise on May 24, 2019.