Parallel and distributed algorithms – Examination, Session 1 (2 hours)

February 2022

Exercise 1: Basic knowledge (4 points)

You have to answer as shortly as possible to the following questions.

1. What is the consistent mode in a PRAM CRCW machine?
2. How can you transform the following PRAM CRCW algorithm into a PRAM EREW algorithm?
```
INPUT: T an Array[1...N] of Integer
FOR each PE i∈[2...N] in parallel:
    T[ i ]⟵T[ i−1 ] + T[ i ]
```
3. What is the main problem of the *shared* memory in CUDA?
4. Assuming $(1024, 1024, 64)$ being the maximum numbers of threads a given CUDA device supports for dimensions (x, y, z), what is its maximum number of threads per block?

Exercise 2: Design Patterns for HPC (6 points)

In this exercise we assume the following integer vectors indexed from 0 exist:

- $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$
- $key = [0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3]$
- $map = [8, 0, 9, 1, 10, 2, 11, 3, 12, 4, 13, 5, 14, 6, 15, 7]$

Moreover, the used binary operator is always the integer addition, and **we consider only *exclusive*** $SCAN$. What are the results of the following parallel patterns applied to these vectors?

1. GATHER(A, map)
2. SCATTER(A, map)
3. REDUCE(A)
4. SEGMENTED-REDUCE(A, key)
5. SCAN(A)
6. SEGMENTED-SCAN(A, key)

Exercise 3: Normalization ($\sim$ 10 points)

In this exercise we consider as input a RGB image, and as output the input image after normalization. This normalization uses a very simple linear filter such that each color is multiplied by a factor $N = 255.0/M$ where `uchar` M is the maximum value of all the pixels' components Red Green and Blue. Notice that the maximum is made for all components, meaning it is maximum of the per channel maximum ($M = max(max(R), max(G), max(B))$). Hence, to do this normalization you have to compute the maximum value of the input image.

1. Write a first implementation using Thrust. The function to fill is:
```
void imageNormalizeThrust(
    thrust::device_vector<uchar3>& input,
    thrust::device_vector<uchar3>& output
);
```
2. Write a second implementation using Cuda. The function to fill is:
```
void imageNormalizeThrust(
    uchar3 const*const d_input,
    uchar3      *const d_output
);
```

These functions use device pointers, so there is no need to move data to device (it is already onto the device!). Notice that the second CUDA implementation should be optimized using what you saw during the lectures (shared memory, synchronization and so on).