UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Benjamin Resch (benjamin.resch@uni-tuebingen.de)
Fabian Groh (fabian.groh@uni-tuebingen.de)
Raphael Braun (raphael.braun@uni-tuebingen.de)

2. März 2017

Massively Parallel Computing
Assignment 4

Submission deadline for the exercises: 5. März 2017

Hints

Download the framework `exercise04.tar.gz` from the Ilias course web page.
Present your results to the exercise instructors to get a grading on this exercise sheet.

4.1 Cell Coverage (100 P)

In a large simplified cell phone network a number of senders have been placed randomly, each radiating at a different power level. The radiation power decreases with the squared distance. In our setup, a phone can still operate if it receives more than 1 unit.
Your task is to implement a system that, given the set of senders and a random collection of receivers, determines how many receivers will not be covered by the current system. The number of receivers might be large (e.g. 1.000.000) and the number of senders in the range of (1.000 - 100.000).

Work Description

- In the provided skeleton you will find code that already sets up the sender and receiver arrays. Furthermore, code for checking for coverage is already provided, as well as a naive CPU and Cuda implementation for the problem.
- In order to make the system run fast you have to do the following:
  - sort the senders into a 2D array of buckets
  - sort the receivers into a 2D array of buckets
- Steps:
  - Scan for bucket sizes (40 P)
  - Calculate buckets' start positions (20 P)
  - Sort the sender/receiver positions according to bucket boundaries (40 P)
- Now the system can easily determine the nearest sender for each receiver by looking in just a small neighborhood of sender buckets around the receiver's bucket. This way the complexity is significantly reduced from O(MN) to roughly O(M). The entire system is implemented in `calculateSignalStrengthsSortedCuda()`.
- The function `calculateSignalStrengthsSortedKernel()` will use the result of your sorting.
- We chose bucket sort for its speed, simplicity and because of memory constraints.