Assignment Sheet 4

Computing the coverage in a cell phone network

Todo

- **Download exercise04.tar.gz from course web page.**

```
tar –xzf exercise04.tar.gz
```

Cell Coverage

- **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. 10.000.000) 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 naïve 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
- **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.**

Hints

- **The order of the elements in each bucket is not significant. In the bucket sort you do not need to sort the elements within each bucket, rather distribute the elements to their respective buckets.**
- **You might want to use atomic functions to count how many elements will end up in each bucket.**
- **The problem size requires you to store the partially sorted elements in an array of the same size as the input array. In order to compute where to start each bucket you would need a PrefixSum. As we have a relatively small number of buckets, it would be o.k. to calculate the necessary PrefixSum sequentially in a single thread on the GPU, avoiding any up and download:** `kernel<<<1,1>>>()`