UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl für Computergrafik
Lukas Ruppert (lukas.ruppert@uni-tuebingen.de)
Mehran Hosseinzadeh (mehran.hosseinzadeh@uni-tuebingen.de)
Deborah Kornwolf (deborah.kornwolf@student.uni-tuebingen.de)
Philipp Langsteiner (philipp.langsteiner@student.uni-tuebingen.de)

February 28, 2025

Massively Parallel Computing
Assignment 3

Instructions

Download the framework `exercise03.zip` from the ILIAS course web page.

Present your results to the exercise instructors to get a grading on this exercise sheet.

3.1 Reduction (50)

Your task is to implement another variant of the dot product program.

- Start from the `reduction` pattern.
- The actual `dotProdKernel` works just like before: It converts two vectors of arbitrary dimension to a vector of `MAX_BLOCKS * MAX_THREADS` numbers which have to be summed up to get the final result.
- Instead of calculating this sum on the CPU, reduce the values on GPU by implementing the `reduceSumKernel`.
  This kernel should always reduce `gridDim.x * blockDim.x` elements to `gridDim.x` elements (each block should output one value).
  Run this reduction in shared memory!
- Call this kernel twice:
  - 1st call: Reduce the `MAX_BLOCKS * MAX_THREADS` elements resulting from the `dotProdKernel` call to `MAX_BLOCKS` elements.
  - 2nd call: Reduce the result from the first pass further to one element by running `MAX_BLOCKS` threads and only one block.

3.2 Compaction (50)

Sometimes the number of output items per thread needs to be variable. In this case, parallel threads do not know to which location to write, because this depends on the number of items generated by the previous threads.

The scan (prefix sum) algorithm solves the problem by generating an index array of the correct locations.

- Start with the `prefix` framework. This framework segments an image into Voronoi cells which are defined by local maxima in the image.
  The local maximum detector marks all found maximas in an integer `gpuFeatureImg` (one element per pixel) with a `1`. The Voronoi image generator instead needs a list with the pixel indices of all features.
- Your task is to implement the following on GPU:

  - Calculate a prefix sum from the `gpuFeatureImg`.
  - Use the prefix sum to assemble a `gpuFeatureList` which can be used by the Voronoi image generator.

- The maximum detection, Voronoi cell generation and a CPU version of the `gpuFeatureList` generation are already included.

- Because of the important parallel computing pattern of the scan, this is a highly recommended exercise.

- Use the provided skeleton and fill in the missing gaps

- Use the `compute-sanitizer` to check for memory-access problems within your kernels.

- Data
  - In folder `images/` you will find example input images
  - In folder `referenceImages/` we have pre-computed the solution for different input images
  - Use these pre-computed solutions and the CPU reference to check that your code works correctly!