GPU Programming 2020/21

Assignment 2

Due: 30/11/2020, 23:59

- Each student has to implement and submit the solution to the assignment her-/himself (plagiarism will be punished according to the study regulations).
- Base your solution on the skeleton code that is provided.
- Submit your implementation as well as the plot(s) before the deadline on the course site. Your code has to compile and run with the given CMake file on the machines in G29-426.

In this assignment we will implement a parallel version of Mergesort using C++ threads.

The divide step of the algorithm parallelizes naturally through the tree structure of the Divide & Conquer algorithm. However, performing the merging serially at each tree node of the up-sweep tree leads to sub-optimal parallel performance and the usual merging strategy cannot be parallelized easily. To obtain good parallel performance, we will hence replace the serial merge by the following Divide & Conquer strategy (that is nested into the Divide & Conquer of Mergesort).

At each node of the up-sweep tree of Mergesort, i.e. whenever two arrays have to be merged, we select the longer array of A and B, say A. We then take as pivot its element at $\lfloor \text{len}(A)/2 \rfloor$ and split the array there into A_1 and A_2. The second array, B, is split where the pivot would be inserted, yielding B_1 and B_2. We thus have two times two arrays with the partial ordering $\{A_1, B_1\} < \{A_2, B_2\}$. We perform the pivot selection and subdivision steps again on the sub-arrays A_1, B_1 and A_2, B_2. This yields a binary tree. When we reach the leafs of this tree (or sufficiently small arrays in practice) then merging is trivial. Through the pivot selection and splitting that was performed in the down-sweep, the up-sweep (or conquer) part of the algorithm yields directly a completely sorted array that is the merger of A and B.[1] We will show later in the course that this algorithm yields very good parallel performance with a sufficient number of threads.

[1] You should make sure you understood the algorithm by performing it by hand for two small input arrays A and B.

Your tasks are as follows:

1.) Download the skeleton code and generate the build system using `cmake`.

2.) Write a serial version of Mergesort that uses at most $O(n)$ additional memory. Verify your implementation by comparing to `std::sort()`.

3.) Implement a parallel version of Mergesort for a variable number of threads that exploits the tree structure of Mergesort (but still uses the classical merging algorithm). (5 Points)

4.) Implement a function `mergeParallel()` that implements the Divide & Conquer version of merge discussed above. Integrate the function into your parallel Mergesort implementation. (5 Points)

5.) Benchmark again with both of your parallel implementations and your serial for arrays of size 2^k with $k = 8 \cdots 14$. Also include again `std::sort()` into your comparison. For your parallel code, explore the different possibilities how threads can be allocated to the parallelization of Mergesort and the parallel merge. Generate a plot (or plots) that show(s) the results of these experiments. (5 Points)