PS 703106
Exercise 8

Group 4:
• Jonas Boutelhik
• Michael Thöni
• Thomas Urban

Hardware:
Intel i7-4710HQ was used for all calulations (Sequential, openMP and openCL)
Processor Base Frequency: 2.50 GHz (Quad-Core)
Processor Graphics:
HD Graphics 4600

Implement an OpenCL version of the prefix sum algorithm (second step of the count sort algorithm) as discussed in the lecture. Additional, a sequential test program has been provided to check the result of the parallel progams.

o Prefix sum implementation for a single work group according to Hillis and Steele (hillissteele.{c/cl}).

Hillis & Steele is a paralles scan algorithm with a single workgroup which requires two buffers of length n. Fig. 1 shows the algorithm for $n=8=2^3$. The two buffers are read and written alternately (toggle) from step to step to avoid collision. The complexity is O(n log(n)).

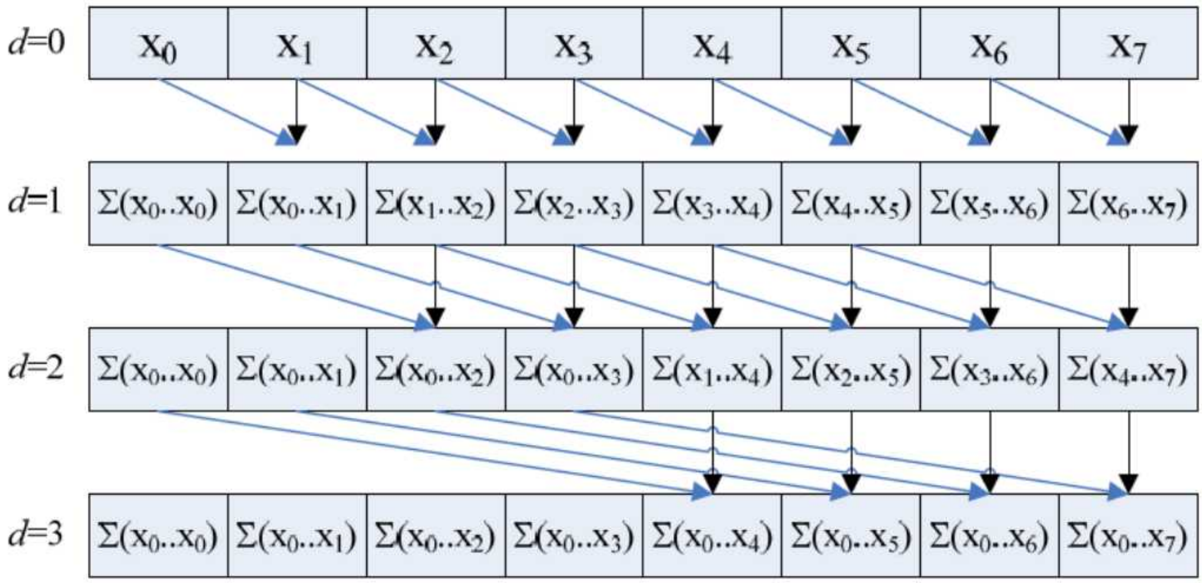

Figure 1: Parallel Scan Algorithm Hillis & Steele (1986) with 3 steps (2^3)

o Optimized implementation using down-sweep step (downsweep.{c/cl}).

Build a balanced binary tree on the input data and sweep it to and then from the root improves the algorithm. The first step, as shown in Fig. 2, is called up-sweep and raverses from the leaves to the root building a partial sums at internal nodes in the tree. Traversing back the tree building the scan from the partial sums. This step is called down-sweep and is shown in Fig. 3. This algorithm as well as Hillis & Steel have to be performed with only one working-group.

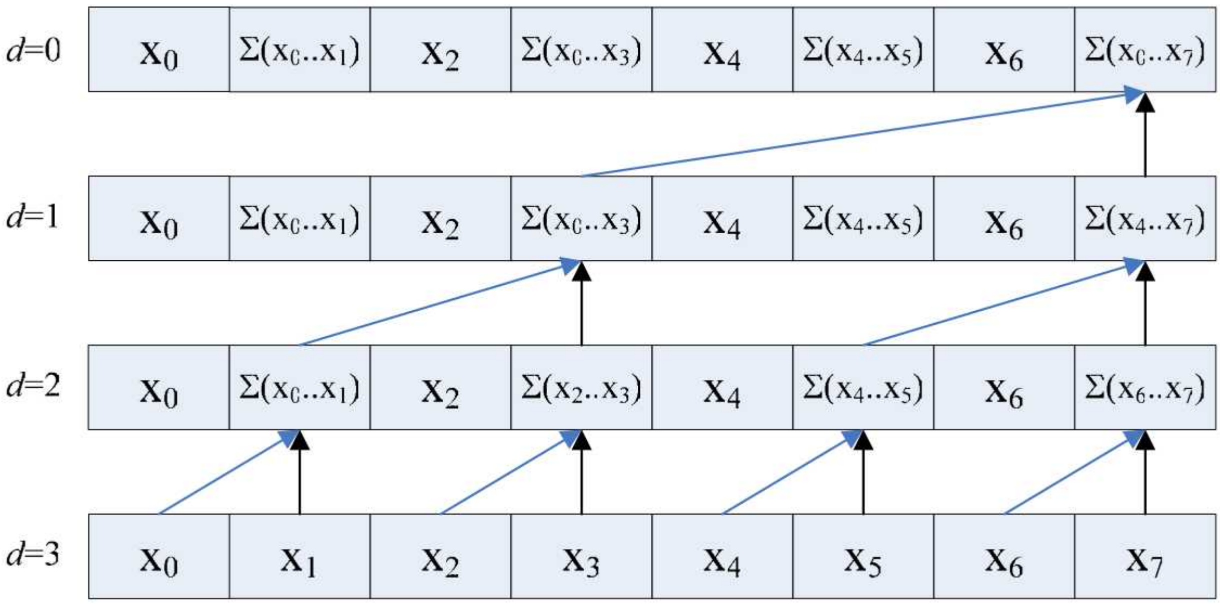

Figure 2: An illustration of the up-sweep, or reduce, phase of a work-efficient sum scan algorithm.

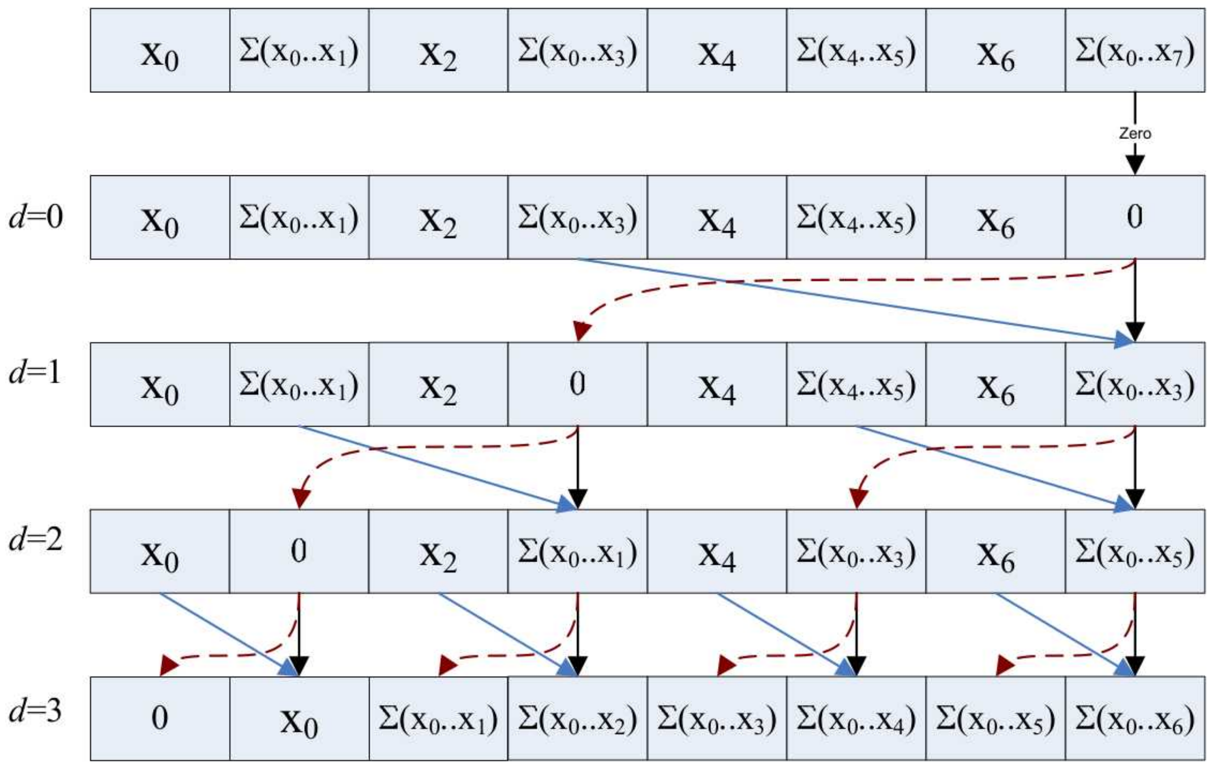

Figure 3: An illustration of the down-sweep phase of the work-efficient parallel sum scan algorithm. Notice that the first step zeros the last element of the array.

o Extension to arbitrarily large arrays using multiple work groups
(prefixglobal.{c/cl}).

The last progam is the optimization of the down-sweep algorithm. To use more working-groups padding is used to eliminate bank conflicts as seen in Fig. 4.
After this optimization step the array can be devided and for the sum scan every part of array can be calculated within a kernal call. The sum of the previous array-parts have to be added as seen in Fig. 5.

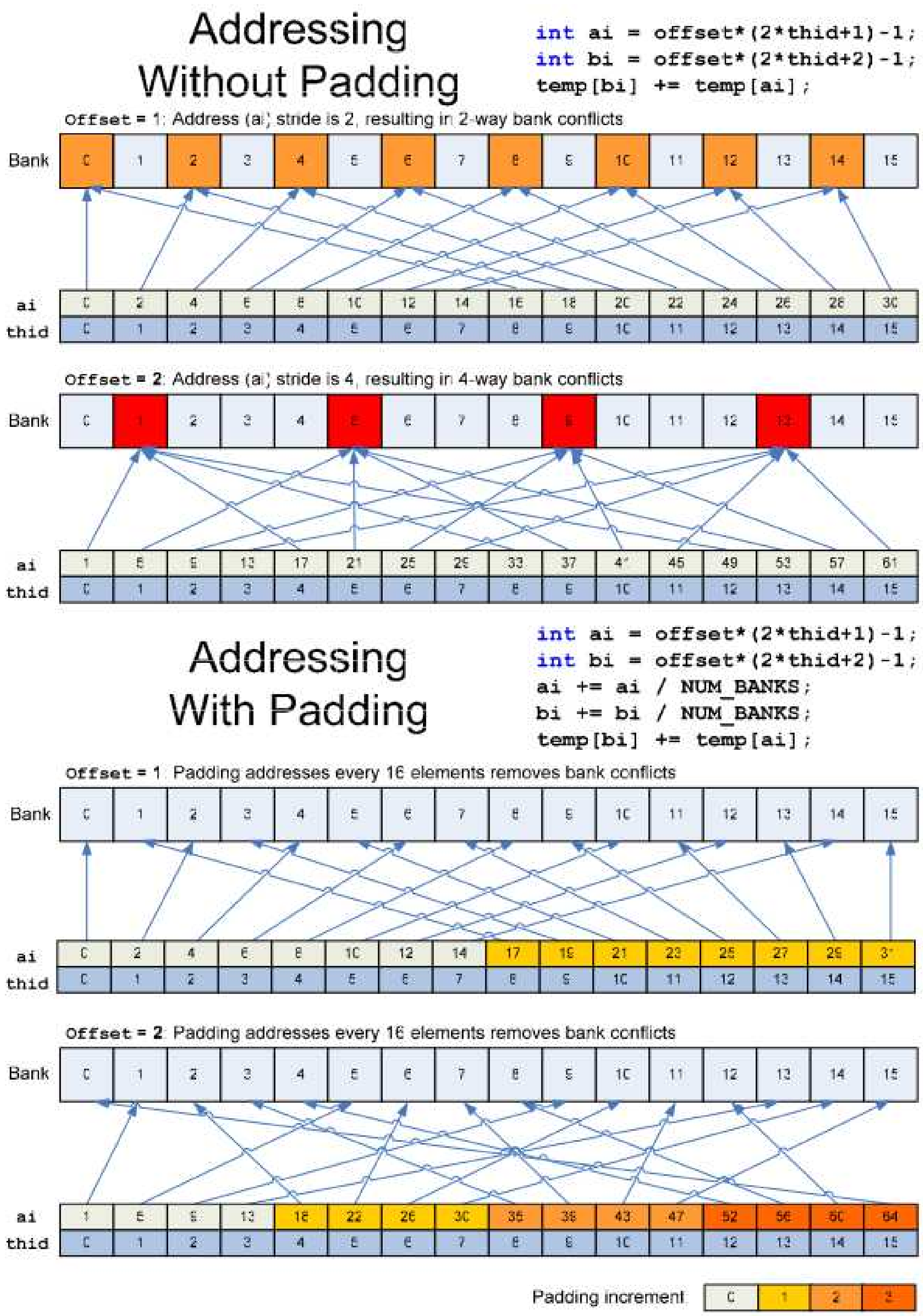

Figure 4: Simple padding applied to shared memory addresses can eliminate high-degree bank conflicts during tree-based algorithms like scan. The top of the diagram shows addressing without padding and the resulting bank conflicts. The bottom shows padded addressing with zero bank conflicts.

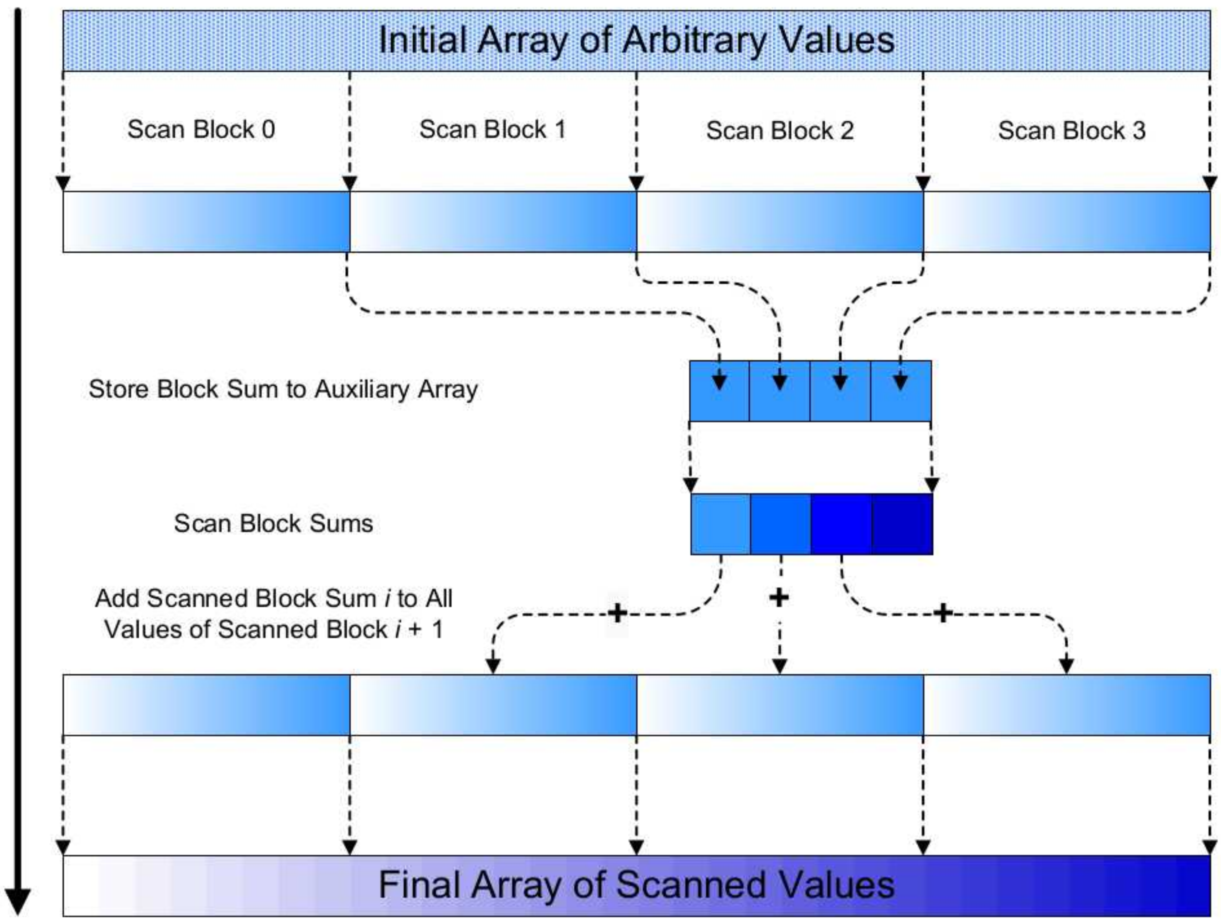

Figure 5: Algorithm for performing a sum scan on a large array of values.

The last Figure 6 visualizes the kernel run time of the 3 different algorithms with 128, 512, 1024, 2048 and 4096 elements to be prefix-sumed.

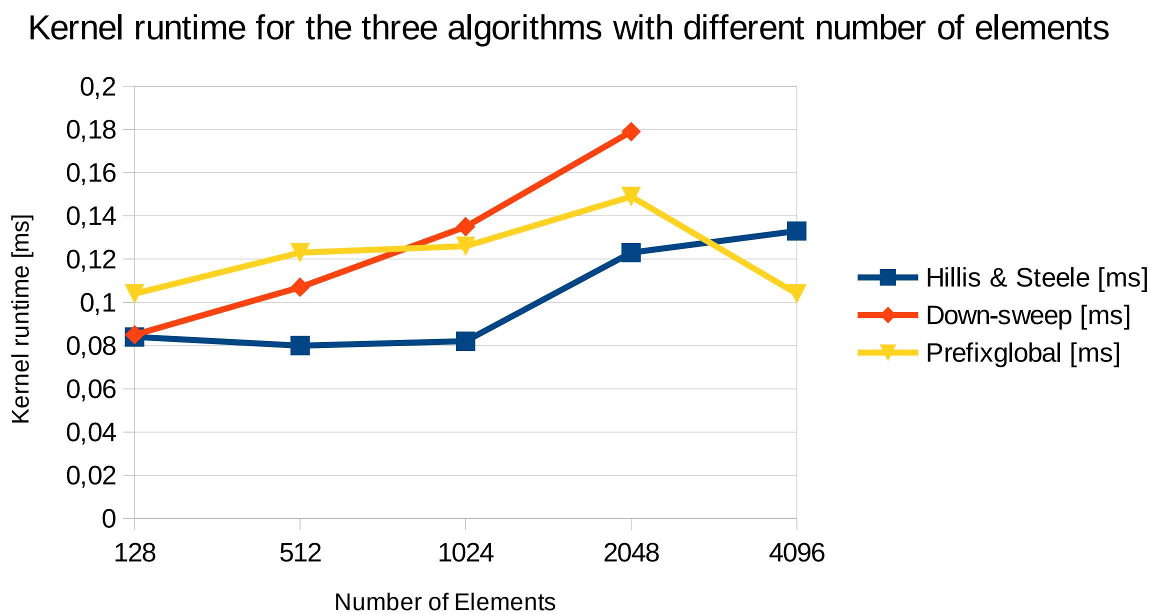

Fig. 6: Kernel runtimes for the 3 algorithms with a different number of elements

Contrary to expectations, the Hillis & Steele algorithm shows the best result. Maybe the down-sweep algorithm will beat the other with increasing number of elements. Unfortunately we were not able to test more elements because the memory was limited.