Exploring and Demonstrating Halide's Optimization Space for Fusing Stencils

This project refers to exploring and demonstrating by CUDA code written by you some of the optimization recipes proposed by Halide [1] in the context of stencil fusion (image processing pipelines). Halide's github repository is here. Some slides related to Halide are also provided, but we encourage you to read the paper.

[1] Jonathan Ragan-Kelly, Connelly Barnes and Andrew Adams, "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines", In Procs. of Int. Conf. Programming Language Design and Implementation (PLDI), 2013.

The task of this project is to implement in CUDA various optimizations recipes proposed for optimizing stencil fusion. You should probably start with the "blur" simple example presented in the paper and implement the five optimization recipes that are shown in the figure depicting the optimizations pace, i.e., the three extreme cases + the two optimized ones (overlapped tiling + sliding window within tiles).

After this warm-up, you should move to optimize "iterative" fusion of two-dimensional and three-dimensional stencils. By iterative, we mean that you fuse the same stencil computation once or two (multiple) times; this simulate a stencil being executed inside a loop, hence the name "iterative".

On GPUs, one would predict that 2D stencil fusion is best accomplished with overlapped tiling and 3D stencil fusion needs some sort of sliding window (probably within a tile). Try to develop and implement as many optimization recipes you can, while having time to systematically analyze the performance of each and compare against other recipes used for the same problem.

When examining the performance use memory throughput (GB/sec) as performance metric (why?). As a number of bytes accessed from global memory, you should use the minimal number of accesses: one can reason that since each element of the input array to fusion must be read and each element of the final result (array) of fusion must be written then the minimal number of bytes corresponds to the sum between the sizes of the input and result arrays of the whole fused computation.

For simplicity of implementation, it is allowed (and encouraged) to consider that a stencil computation shrinks each dimension with 2*R, where R is the radius. For example, if a 2-dimensional input is of size [N+4*R, N+4*R], the intermediate result of the first stencil is [N+2*R, N+2*R] and the second stencil's result is [N,N]. This eliminates the branches that would otherwise be necessary to treat boundary conditions, thus leading to simpler code.

Your report should contain at least:

1. a summary of the Halide paper that presents (1) the optimizations space organized on three axes (amount of exploited parallelism, locality, redundant computation), and (2) several of the code transformations (optimization recipes) that navigate this space.
2. The rationale for selecting the examples of stencil fusion that you have decided to analyze, together with a high-level description of each one of them, i.e., where does it fit within the optimization space (try to place it on the figure depicting the optimization space). As well, try to summarize for each studied example the amount of redundant computation, parallelism and locality (see the table in the paper).
3. An presentation of the CUDA code that implements each example – it is fine for readability to use C-like pseudocode in which you annotate with comments which loops form the grid, which loops form the CUDA block of threads, which loops are sequential, which buffers are allocated in shared memory and which are in global memory.
4. As well, try to explain at least at a high level the re-write rules that produced the code, i.e., reason why the transformation is and why does it preserves the semantics of the original code.
5. Most importantly, a systematic evaluation of performance that demonstrates the extent to which each implementation utilizes the hardware and compares the performance across different optimization recipes of the same stencil program.

Please submit the code as a zip (or tar.gz) archive that can be simply de-archived and in which everything is runnable and reproducible by simply running a “Makefile” that you provide on the futhark servers. Please include in your report how to reproduce the presented results --- ideally this is achieved just by typing “make”.