

Basic Setup

Follow the instructions for class account setup found on <http://www.ece.lsu.edu/gp/proc.html>. This assignment uses code in the `hw04` directory, file `hw04.cu` is to be modified.

If the class account has been set up properly, the code can be built from within Emacs by pressing `F9` when visiting any file in the `../hw04` directory or when in an Emacs shell buffer (which can be entered using `Alt-x shell Enter`). The code can be built from the command line using the command `make -j 4` (assuming `../hw04` is the current directory). The makefile builds three versions of each program, named `hw04`, `hw04-debug`, and `hw04-cuda-debug`. The versions with the `-debug` suffix are compiled with host optimization turned off, which facilitates debugging. The `hw04-cuda-debug` version is compiled for CUDA kernel debugging, using `cuda-gdb`. It runs without performance data (such as the data plotted under `I/op`) and is not optimized. Use `hw04-cuda-debug` when using `cuda-gdb` to debug kernel code. Host code can be debugged that way too, but execution is faster when `gdb` is used on `hw04-debug`.

Quickly check whether the build is successful with the command `./hw04`.

The makefile will compile code for a GPU on the system it was run, favoring the GPU that's not connected to a display. Re-run `make` when moving to a different system. The Makefile should automatically detect whether the GPU for which the executable was built matches the GPU on the current system, and re-build if needed.

For this assignment edit code in `hw04-kernel.cu`.

Using hw04

Without any command-line arguments `hw04` will run the radix sort for three different radices each on five block sizes, choosing the number of blocks needed so that 32 warps will be resident per SM.

Problem 1: The values shown under the second I/W for high-radix, small block runs look too large. Perhaps some activity done by the Pass 2 kernel is not being taken into account. Fix the value shown under the second I/W column so that it correctly shows the number of executed instructions per unit work.

To do so compute the amount of work per thread based on the tile size, radix and other factors and use that value in the code. Do this for the original code.

While solving this problem you may discover that some parts of the pass 2 kernel were inefficiently written. Take this inefficiency in to account, but don't try to fix the problem.

Problem 2: The performance of pass 1 is determined by the 1-bit split routine, and 1-bit split frequently computes prefix sums. Modify the 1-bit split routine so that calls `syncthreads` fewer times. Do so by computing a prefix for each warp (which can avoid `syncthreads`), then compute a prefix of the sums in each warp. Try doing so using the ballot technique used in `vtx-xform-sparse`. Also try using `sync_shfl_up` within a warp.