# EECS 368/468 – Lab3
# Programming Massively Parallel Processors with CUDA

## *Optimizing Histograms in CUDA*

In this assignment, you will work with your teammates to write and optimize a histogram code in CUDA.

## Introduction

Histograms are a commonly used analysis tool in many application domains, including image processing and data mining. They show the frequency of occurrence of data elements over discrete intervals, also known as bins. A simple example for the use of histograms is determining the distribution of a set of grades. Say you have a record of grades: 0, 1, 1, 4, 0, 2, 5, 5, 5. The above grades ranging from 0 to 5 result in the following 6-bin histogram, with each bin representing one grade: 2, 2, 1, 0, 1, 3.

## Description of Lab3 files

| | |
|---|---|
| `Makefile:` | The makefile to compile your code. |
| `util.h:` | Header file with some utility macros and function prototypes. |
| `util.c:` | Source file with some utility functions. |
| `ref_2dhisto.h:` | Header file for the reference kernel. |
| `ref_2dhisto.cpp:` | Source file for the scalar reference implementation of the kernel. |
| `test_harness.cpp:` | Source file with `main()` that has sample calls to the kernels. |
| `opt_2dhisto.h:` | Header file for the parallel optimized kernel (currently empty). |
| `opt_2dhisto.cu:` | Source file for the parallel optimized kernel (currently empty). |

## Install Lab3 at your Wilkinson Lab account

1. Download the lab3 tarball from canvas into the `EECS468-CUDA-Labs` directory in your Wilkinson Lab account.
2. **Remember**: every time you login to the Wilkinson lab to program in CUDA you need to setup the CUDA environment. If you use csh or tcsh:
   ```
   source /usr/local/cuda-5.0/cuda-env.csh
   ```
   or if you use the bash shell:
   ```
   . /usr/local/cuda-5.0/cuda-env.sh
   ```
3. Install the lab3 tarball and compile the lab sources:
   ```
   cd EECS468-CUDA-Labs
   tar xvf EECS468-CUDA-Lab3.tgz
   make
   ```

## Complete the assignment

The source code you will be working on is at `EECS468-CUDA-Labs/labs/src/lab3`. Compiling your code will produce the executable `lab3` in the directory `EECS468-CUDA-Labs/labs/bin/linux/release`.

There are two modes of operation for the application:

a. **No arguments**: The application will use a default seed value for the random number generator when creating the input image.
b. **One argument**: The application will use the seed value provided as a command-line argument. When measuring the performance of your application, we will use this mode with a set of different seed values.

The application reports the timing information for the sequential code followed by the timing information of the parallel implementation. It will also compare the two outputs and print "Test PASSED" if they are identical, and "Test FAILED" otherwise. The base code provided should compile and run without errors or warnings, but will fail the comparison.

Your task is to implement an optimized function `void opt_2dhisto()`. I expect that you will **devise optimizations to try beyond what we described in the class lecture**. For reference, the method `ref_2dhisto()` constructs a histogram from the bin IDs passed in "`input`".

a. "`input`" is a 2D array of input data. These will all be valid bin IDs, so no range checking is required.
b. "`height`" and "`width`" are the height and width of the input.
c. "`bins`" is the histogram. `HISTO_HEIGHT` and `HISTO_WIDTH` are the dimensions of the histogram (the default values are `1` and `1024` respectively, i.e., it is a 1K-bin histogram).

## Assumptions and Constraints

a. The "`input`" data consist of index values into the "`bins`".
b. The "`input`" bins are not uniformly distributed (which makes it an interesting GPU problem).
c. For each bin in the histogram, once the count reaches 255, no further incrementing should occur. This is sometimes called a "saturating counter". DO NOT ROLL OVER.

The scaffolding code is provided for your reference only. You can modify any file you want, or write new code from scratch. It is strongly recommended, however, to use the scaffold provided, as it can save you significant time. If you choose to write your own code, the reference histogram implementation (implemented in files `ref_2dhisto.*`) and the input generation (implemented in `generate_histogram_bins()`, `next_bin()`, and the input dimensionality and seeding in `test_harness.cpp`) should remain the same. The discussion that follows assumes you will use the code provided in the lab tarball.

You should edit the following files: `opt_2dhisto.h`, `opt_2dhisto.cu`, and `test_harness.cpp`. Moreover, it is sufficient to modify the file `test_harness.cpp` only where instructed to do so (view the comments in the file). You should measure the runtime of the kernel execution only; any GPU allocations and data transfers should be done outside the function `opt_2dhisto()`. The arguments to the function `opt_2dhisto()` have been intentionally left out for you to specify based on your implementation. There is no need to modify any other files.

Please note that the file `test_harness.cpp` will be compiled with `gcc` and not `nvcc`. If you insert CUDA calls in it, compilation will fail. The best way to add your CUDA-specific code (for example to allocate device memory or to free device memory) is to implement corresponding functions in the `opt_2dhisto.cu` file, and call these functions from `test_harness.cpp`. This way, your entire implementation will be contained within the `opt_2dhisto.cu` file, making it easy to read and grade.

You may not use anyone else's solution; however, you are allowed to use third-party implementations of **primitive** operations in your solution. If you choose to do so, it is your responsibility to include these libraries in the tarball you submit, and modify the `Makefile` so that your code compiles and runs. You must also **mention any use** of third-party code in your report. Failure to do so will be considered plagiarism. If you are uncertain whether a function is considered a primitive operation or not, please inquire the instructor about it.

## Submitting your solution

1. Create a tarball of your solution, which includes the contents of the `lab3` source folder, with all the changes and additions you made to the source files. Please make sure you do a "`make clobber`" before submitting; do not include object code or executables, as these are typically large files:

   ```
   cd EECS468-CUDA-Labs/labs/src/lab3
   make clobber
   tar cvfz EECS468-lab3-YourNames.tgz *
   ```

   where you should replace "`YourNames`" with the concatenated full names of the team members. Submit your solution tarball via canvas. Only one student in a team needs to submit. If you submit multiple times, only the latest submission will be graded. Please do all submissions from the same account.

2. In addition to the code, add a report file in your `lab3` source directory. Your report should contain a journal of all optimizations you tried, including those that were ultimately abandoned, or worsened the performance, or did not work. Your report should have an entry for every optimization tried, and each entry should briefly note:

   a. What is the goal for the optimization, and briefly describe the changes you made in the source files for that optimization.
   b. Any difficulties with completing the optimization correctly.
   c. The man-hours spent developing the optimization (even if it was abandoned or not working). This will be an indication of the optimization's difficulty.
   d. If finished and working, the speedup of the code after the optimization was applied.

   Please be brief in the descriptions above; there is no need for lengthy descriptions. Even bulleted lists are enough if they convey everything you want to say.

## Grading

Your submission will be graded based on the following parameters.

a. **Correctness** (15%): the kernel produces correct results at the output.
b. **Optimizations** (70%): for this portion, we will grade the thoughtfulness you put into speeding up the application.
c. **Report** (15%): complete and accurate journal. We will at least check for discrepancies, optimizations that you did but didn't report, etc.

Good luck!