MIT EECS 6.815/6.865: Assignment 9:

# Halide on your own!

Due Wednesday December 2 at 9pm

Now that you master the essentials Halide building blocks, let us code a complete image processing pipeline that involves several computation stages. This should give you the opportunity to leverage Halide's scheduling tools and optimize the computation across different stages. This pset will be less guided that the previous one.

The grad version also asks you to implement a simple auto-tuning algorithm to explore various schedules and find the most efficient automatically.

## 1 Summary

- Implementing the Harris Corner detector in Halide
- Scheduling the computation pipeline for Harris' corner detector
- 6.865: Writing an auto-tuner to find a good schedule

## 2 Harris corners detector

We provide you with an reference C++ implementation of the Harris corner detector you already implemented in the Panorama pset. You can find the code in `reference_implementation.cpp`.

Following this code, implement the same algorithm in Halide in `a9.cpp`, and test it using `a9_main.cpp`. While developping the first version of the algorithm, you can test it on a low resolution image (e.g. `hk-small.png`). Make sure to test the intermediate steps as well!

You can output the intermediate steps of the reference algorithms by uncommenting code in `reference_implementation.cpp`. But make sure that you are not outputting images in the reference function when comes the time to compare performances: writing `.png` files to disk takes a significant chunk of time!

Once you have a working implementation of the filter, run it with a default schedule in which all your `Func` are `compute_root` (you can use `apply_auto_schedule` for this). Then come up with a better schedule for Harris, leverage the Halide primitives as best as you can! You might want to work on higher resolution imagery for this (like `hk.png`) since the effect of varying the tile size in parallel loops for example might be less significant when there is little computation to do in the first place.

For time measurement, please also provide your machine's characteristics.

1.a Implement `HarrisCorners` in `a9.cpp`. The final output should match that of the reference code (everywhere but perhaps not at the image boundaries).

1.b What are the running time and throughput of the reference implementation on your machine?

1.c What are the running time and throughput of your Halide implementation with the default schedule?

1.d Find a better schedule than the default one and report your performances (running time and throughput, as well as you machine's characteristics).

## 3 6.865 only: Auto-tuning

This is an open-ended part. Write C++ code that systematically tries many schedules for your Harris corner detector. Automatically explore different scheduling strategies (e.g. what gets tiled, what gets computed at what granularity) and different numerical values for things such as tile size. Tutorial 10 from the previous pset might give you some idea but it is relatively simplistic. In particular, note that some strategy decision (e.g. tiling a partic- ular computation) can generate new possible choices (tile size) that might not be relevant for other strategies.

Have a look at the `a9_main.cpp`, the function `autoschedule_harris` gives and example on how you may proceed to test your various schedules. Note that the `HarrisCorners` function you implemented takes `schedule_index` and `schedule_parameters` as optional arguments. You can use this to dynamically select the schedule in `HarrisCorners` and pass an array of integer parameters needed by that schedule, if any.

2 Describe your overall approach in one paragraph, turn in your code and your best schedule, together with its performance.

## 4 Submission

Turn in your files to the online submission system (link is on Stellar) and make sure all your files are in the `asst` directory under the root of the zip file. If your code compiles on the submission system, it is organized correctly. The submission system will run code in your main function, but we will not use this code for grading. The submission system should also show you the image your code writes to the `./Output` directory

In the submission system, there will be a form in which you should answer the following questions:

- How long did the assignment take? (in minutes)
- Potential issues with your solution and explanation of partial completion (for partial credit)
- Any extra credit you may have implemented and their function signatures if applicable
- Collaboration acknowledgment (you must write your own code)
- What was most unclear/difficult?
- What was most exciting?