MIT EECS 6.815/6.865: Assignment 8:

Introduction to Halide and paper review

Due Wednesday Novemeber 18 at 9pm

1 Summary

In this pset, you will get acquainted with Halide. We will first go through environment setup. Then you will follow a tutorial that introduces the basic elements of the language and apply them on simple exercises.

The description of the tutorials exercises can be found in the `tutorial*.cpp` files. Go through the tutorials one by one, each tutorial is supposed to explain basic concepts and test them on slightly different use cases.

The grad version has a paper review in addition of the Halide tutorials and exercises.

- Setting up Halide
- Tutorials on the main concepts of the language
- Application Exercises
- 6.865: paper review

2 Installing Halide

The next two problem sets will use Halide. If you have trouble setting it up, please come to office hours and ask us early on.

2.1 Obtaining Halide

Precompiled binaries. We recommend that you use precompiled binaries of Halide from https://github.com/halide/Halide/releases. Use the trunk version for your favourite operating system. Linux will be easiest, followed closely by Mac. Windows should be more troublesome, but not impossible. For Linux, ake sure to get the binaries corresponding to your version of GCC and 32/64 bits depending on your OS.

We strongly recommend Windows users use Athena for the Halide problem sets.

Compiled from source. If the precompiled binaries do not work right away, you can compile Halide from source, which is what most of us who actually use Halide have done. You will find instructions at https://github.com/halide/Halide (see "Building Halide"). But try the precompiled binaries first.

External libraries (Mac users). You need to or must have installed libpng through Macports or Homebrew. Assuming that the installation succeeded, you should be able to run

```
libpng-config --I_opts
libpng-config --L_opts
```

Please add the output of the above commands to the variables `PNG_INC` and `PNG_LIB` in the Makefile.

2.2 Working on Athena

Here is a step-by-step on working on Athena for Windows users, or anyone who wishes to.

- log in to Athena, e.g.: `ssh username@athena.dialup.mit.edu`.
- check the version of gcc installed: `gcc -v`, should return something like: `gcc version 4.8.4`.
- choose a directory to install Halide: `cd $HOME/Documents`
- download the appropriate version of Halide from https://github.com/halide/Halide/releases, e.g. `wget https://github.com/halide/Halide/releases/download/release_2015_10_22/halide-linux-64-gcc48-trunk-c815789464d4d0dc790ac13b084a4da44459983f.tgz`.
- extract the archive `tar -zxf halide.tar.gz` and note the root path of the installation.
- To transfer files from your local machine to Athena you can use `rsync -rv localpath userame@athena.dialup.mit.edu:remotepath`.
- To transfer files from Athena to your local machine you can use `rsync -rv userame@athena.dialup.mit.edu:remotepath localpath`. This can come in handy to pull the `Output` folder after running your code for example.

2.3 Documentation

You can find Halide documentation and examples at https://github.com/halide/Halide/wiki. Get in touch with Michaël if things don't work or post on Piazza!

3 Running the code

By now you must have a Halide installation on your machines. Modify `HALIDE_DIR` in the assignment's Makefile. This should point to the root of the Halide path on your machine.

1.a Change the environment variable (in the Makefile) `HALIDE_DIR` to point to the root of your Halide installation.

1.b OSX users: install `libpng` via Homebrew of Macports and update the environment variables `PNG_INC` and `PNG_LIB` if needed.

1.c Test your installation by running `make`. The code should compile and you will be able to run the tutorials.

In this pset we will not any of the code you wrote for the previous assignments. In particular we will not use the same `Image` class as usual.

All deliverables have to be implemented in a `a8.cpp` in their respective functions. Each function has a description of what-to-do in comments above. Please read the comments. Performance statistics have to be reported on the submission system.

As usual, use `a8_main.cpp` to test and debug your functions.

3.1 Hints

- We have provided a function `profile()` to measure Halide function performance.
- We have provided a function `apply_default_schedule` that sets all Halide functions to `compute_root`. When applicable, your schedules should be faster that this baseline.
- Use `compile_to_lowered_stmt()` to output loop nests for debugging schedules, some tutorials show how it is meant to be used.

4 Func, Var, Expr and input/output buffers

Read on through `tutorial01` to `tutorial04` in the corresponding `.cpp` files.

2.a In `a8.cpp`, implement the function `SmoothGradNormalized`. This function is very similar to the example in `tutorial01`, but the output is normalized by 1024 in the Halide pipeline. This will teach you how to generate a grayscale image.

2.b Implement `WavyRGB` in `a8.cpp`. This will teach you how to generate an RGB image.

2.c Implement `Luminance` in `a8.cpp`. You will learn to read an input

RGB image and produce an output grayscale image.

2.d Implement `Sobel` in `a8.cpp`. You will learn to write a multi-stage pipeline, that uses neighboring pixels.

5 Scheduling a computation

The previous section should have walked you through Halide's basic syntax. You are now able to specify your algorithm in Halide using Func, Var, Expr and Image buffers.

In this section we will learn how to change the *schedule* of a computation: determine which pixel gets computed, in which order. Halide's scheduling syntax allows us to quickly explore various schedules: tiling, parallelizing, vectorizing operations, fusing stages together, etc. This allows us to quickly optimize our algorithm and really is where the language shines.

Read `tutorial05` to get acquainted with the essential scheduling primitives of Halide on a single stage pipeline: `reorder, fuse, split, tile, vectorize, parallelize, unroll`.

For multiple stage pipelines (for example, extract the x and y gradients from an image, get their magnitude and blur the result), Halide's default schedule is to *inline* all the operations. This means that no intermediate buffer will be allocated, and for every pixel in the output, the values in the intermediate stages will be recomputed. The extreme opposite to inlining is when every stage in the pipeline is scheduled as `compute_root`: in this case every stage writes all of the required values to an intermediate buffer. While scheduling Halide algorithms, we try to find the optimal mid-point between these options, balancing locality (i.e. minimizing large memory transfers) with redundancy (i.e. minimizing the extra-work).

To make things more concrete, read on `tutorial06` which schedule a box blur. The algorithm has two stages, first blur along x then along y. You will learn about `compute_at, compute_root, compute_inline`

3.a Report the runtime, throughput of the four schedules in tutorial06. Which is best?

3.b Write equivalent C++ code for schedule 5 of tutorial 06 in `a8.cpp`. (See example for schedule 1 in `a8.cpp`).

3.c Write equivalent C++ code for schedule 6 of tutorial 06 in `a8.cpp`. (See example for schedule 1 in `a8.cpp`).

3.d Write equivalent C++ code for schedule 7 of tutorial 06 in `a8.cpp`. (See example for schedule 1 in `a8.cpp`).

6 Reductions and the select directive

Read `tutorial07` and `tutorial08` to learn about reductions and the select statement.

4 Implement `LocalMax` in `a8.cpp`. You will learn to use the select statement. You can find a reference implementation `LocalMax_cpp` in `reference_implementations.h`.

Read `tutorial09` and `tutorial10` to see how reductions can be used to perform convolutions.

5.a Run tutorial10. What is the best tile width ?

5.b Implement a separable Gaussian blur filtering `Gaussian` in `a8.cpp`. And try to come up with a good schedule for it. This will leverage the skills you learned in the last two tutorials You can find a reference implementation `Gaussian_cpp` in `reference_implementations.h`. (ignore the boundary artifacts)

5.c Implement unsharp masking `Unsharp` in `a8.cpp`. Though you could use the Gaussian Func from the previous question, rewriting the code will give you the opportunity to fuse stages of the pipeline, and potentially get a faster implementation. And try to come up with a good schedule for it.

7 6.865: Paper review

This part is for 6.865 only, and unlike the previous psets, it's going to be all reading and writing. Your job is to review a recent publication in computational photography. This will hopefully give you some more insight into the review process that these papers go through.

You should select one paper from the provided `papers.html` file (if the link is dead, search for the paper on Google), or a computational photography paper from the SIGGRAPH lists here: http://kesen.realtimerendering.com/. If there's another relevant paper that you're particularly interested in reading, ask us about it and we'll probably be okay with it (unless it's something that you're already supposed to read for one of the problem sets). If you're not sure if a particular SIGGRAPH paper falls under computational photography, feel free to ask us on piazza.

You should use the SIGGRAPH review form, available at the following URL: http://s2015.siggraph.org/submitters/technical-papers/review-form. For "Explanation of Rating", try to come up with at least one outstanding question that you think the authors didn't address in the paper. You can skip the "Private Comments". You should put your review in an ascii text file called `review.txt` in the `asst` directory. Try to be thorough! Probably a paragraph or two for each question, and more for "Explanation of Rating".

One final note: don't be compelled to like a paper because it was already published. Every paper has some shortcomings. Maybe the method is very elegant and general but doesn't end up producing very impressive results. Maybe the results are spectacular, but the method itself has some mathematical holes. Maybe everything seems great, but there just aren't enough details to reproduce the results. It's up to you as a reviewer to weigh the strengths and weaknesses of each paper

6 Include your paper review in an ascii text file called `paper_review.txt` and submit it to stellar. See the text for details on what should be included.

8 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?