Project 3: Gaussian Blur Optimization

Computer Architecture I | ShanghaiTech University
Project 2.2 Project 3 Project 4

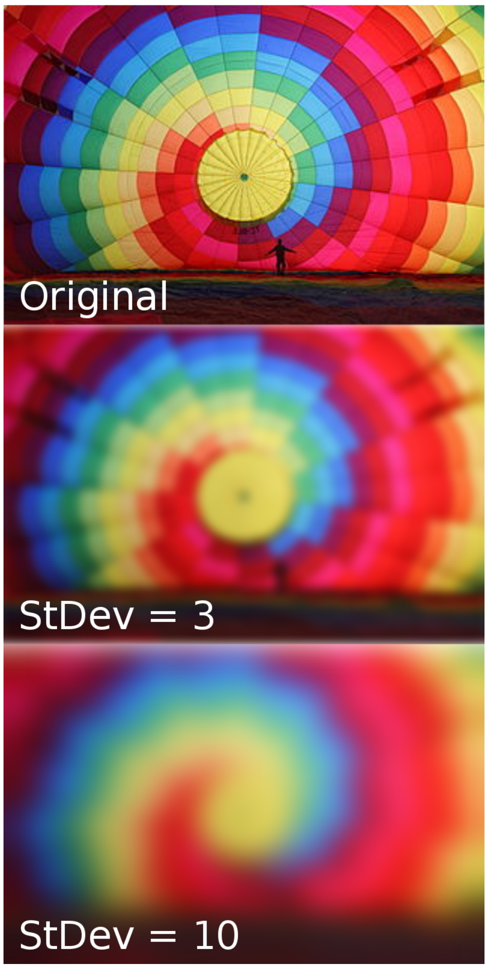
IMPORTANT INFO - PLEASE READ

The projects are part of your design project worth 2 credit points. As such they run in parallel to the actual course. So be aware that the due date for project and homework might be very close to each other! Start early and do not procrastinate.

In this project, we hope you can use all knowledge about computer architecture that your have learnt in this course to optimize the Gaussian Blur algorithm.

Gaussian Blur

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen.



Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function. For more information about Gaussian Blur, you can check it on wiki pedia. Gaussian Blur

In this project, we adopt a 1-dimensional Gaussian distribution kernel with qualities set by the user, and the blurring process is done in two steps: Given image A as our input, we first convolve the kernel over the rows of image A to produce a horizontally blurred image B. We then convolve the kernel over the columns of image B to produce a horizontally and vertically blurred image C. The image C is our final blurred image, and we save it to disk. Image reading and writing is handled by stb_image.h and stb_image_write.h.(We modified some of the codes in stb_image.h and stb_image_write.h for our algorithm, so please use the two files in gitlab repository. Do not use the original ones on github.)

To make a similar bokeh effect(just a little similar), we adapt the Gaussian blur algorithm so that the more one pixel near the edge, the more blur it will be. You can check the basic code we give to you to see how we implement it.

Getting started

Make sure you read through the entire specification before starting the project.

You will be using gitlab to collaborate with your group partner. Autolab will use the files from gitlab. Make sure that you have access to gitlab. In the group CS110_22s_Projects you should have access to your project 3 project. Also, in the group CS110_22s, you should have access to the p3_framework.

Obtain your files

1. Clone your p3 repository from GitLab.
2. In the repository add a remote repo that contains the framework files: `git remote add framework https://autolab.sist.shanghaitech.edu.cn/gitlab/cs110_22s/p3_framework.git`
3. Go and fetch the files: `git fetch framework`
4. Now merge those files with your master branch: `git merge framework/master`
5. The rest of the git commands work as usual.

Files

The framework contains the following files:

- `Makefile` - File which records all dependencies. You can change the CFLAGS and LIBS according to your implementation, but do not change anything in "fast" and "check" as they will be used in grading. Also **do not change the optimization flag -O2**, it could make you get 0 points for this project.
- `README.md` - An introduction file which tells you how to run the program. Remember that the dimension parameters should **both be odd** .
- `gbfloat_base.c` - The reference code for basic Gaussian_blur algorithm.**(You should not change it)**
- `gbfloat_fast.c` - The Gaussian_blur algorithm you need to optimize.
- `stb_image.h` - The library used for image reading.**(You should not change it)**
- `stb_image_write.h` - The library used for image writing.**(You should not change it)**

- `test.jpg` - The reference jpg used for your local testing.
- `test_accuracy.c` - The code you can use to check if two images are the same per pixel(i.e. if all pixels have the same value).

Optimization Techniques

To test the optimization, you may use unnecessary huge Gaussian kernels (size 500+) to test the speed of the program. You can find many obvious optimizations in the implementation we provide, with what you have learnt in Computer Architecture. We are listing some of the possible approaches below:

Algorithm

You may find there are faster algorithms to do Gaussian Blur. You are not allowed to optimize the naive algorithm we give though, as this is not the focus of Computer Architecture. You will receive no point if we find you do that.

Compiler

There are some optimization flags that can be turned on in GCC. The available flags for x86 ISA from GCC is listed here.However, we wish you to do the optimization on your own, instead of relying on the compiler to do it for you. You will receive 0 points if you try to turn on any other optimization flags except for -O2 specified in Makefile.

Multithreading

The first and the easiest approach is to use multithreading to optimize this algorithm, with either `pthread` or `openmp`.

SIMD instructions

Part of this algorithm is also a good candidate for SIMD instructions.

Loop unrolling

Loop unrolling can work very well in combination with SIMD instructions, and you should think about it.

Cache Blocking

Part of this algorithm is also a good candidate for cache blocking.

Grading Policy

Your grade will be divided into two parts:

0. We will first run your code on small test cases on Autolab. If you program produces the correct result and satisfy our time requirement, you receive 40% points.

1. After the due, we will test your code on large test cases. Your grade on this part depends on the execution time of your code. All groups run slower than 1 minute but faster than 2 minutes get 20% points and the fastest 15 groups receive 60% points. The execution time of 15th group is 60% line and 1 minute is 20% line. The grades for the remaining groups are linearly distributed according to their execution time.
2. For your reference, if you only add a single line of #pragma omp parallel for, the running time will be between 1 minute and 2 minutes.
3. If your code should not crash on either stage, or you will receive no point in that stage.
4. Your submission must contain meaningful use of OpenMP(or pthread) and Intel SIMD intrinsics. Otherwise, you will get 0 point from both stages. This check will be done manually after the deadline so there will be no feedback on this from the auto-grader.

There are something that you need to keep in mind: There are some really open-source fast implementations for Gaussian Blur algorithm available. **You should not submit any existing implementations that is not written by you.** But you can refer to some technical reports and articles for the algorithms and optimizations available and implement your own. And make sure after optimization, your code should produce the same image as the original basic code.

Submission

When your project is done, please submit all the files including the framework to your remote GitLab repo by running the following commands.

```
$ git commit -a
$ git push origin master:master
```

Autolab will discard all other files except for `gbfloat_fast.c` and `Makefile`.

How to Autolab

Similar to previous projects, upload your `autolab.txt` to Autolab to submit your project.

Submission Time Announcement

The last time of your submission to the git repo will count towards your submission time - also with respect to slip days. So do not commit to this git after the due date, unless you want to use slip days or are OK with getting fewer points.

Collaborative Coding and Frequent Pushing

You have to work at this project as a team. We invite you to use all of the features of gitlab for your project, for example branches, issues, wiki, milestones, etc.

We require you to push very frequently to gitlab. In your commits we want to see how the code evolved. We do NOT want to see the working code suddenly appear - this will make us suspicious.

We also require that all group members do substantial contributions to the project. This also means that one group member should not finish the project all by himself, but distribute the work among all group members!

At the end of Project 3 we will interview all group members and discuss their contributions, to see if we need to modify the score for certain group members.

Server Configurations

Autolab Server (Just for your reference):

- CPUs: 2 * Intel Xeon E5-2690 v4 2.6 GHz Details here
- Memory: 200GB 4-channel DDR4 (shared by all student tasks)
- **4 threads** are allowed for each grading job

Star Lab Server (Used for final speed test):

- CPU: Intel Xeon E5-2650 v3 2.3 GHz, 10 cores (20 threads) Details here
  - L1 cache 64KB per core
  - L2 cache 256KB per core
  - L3 cache 25600KB
- Memory: 32GB

Schwertfeger, Sören <soerensch *AT* shanghaitech.edu.cn>
Chundong Wang <wangchd *AT* shanghaitech.edu.cn>
Xu, Qing <xuqing2 *AT* shanghaitech.edu.cn>
Yang, Hongdi <yanghd *AT* shanghaitech.edu.cn>
Last modified: 2022-03-26