Parallel Image Processing with CUDA

Austin Bond , Thomas (TJ) Davies , Andrew Fleming , and Justin Newman

James Madison University Department of Computer Science, Harrisonburg, VA 22801, United States

Github Repository, Trello Board

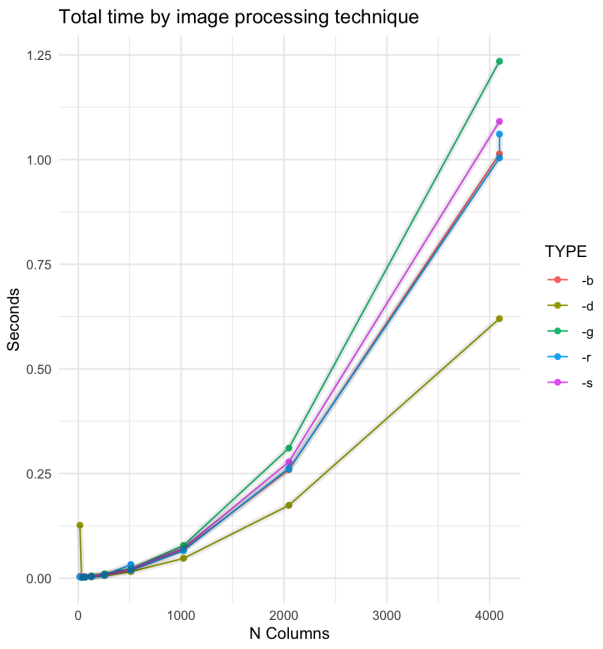
---

1. How do you feel the project is going so far in general? 4.5
2. How urgently do you need feedback on this deliverable? Normal
3. How effectively is your group finding meeting times and getting work done? Great
4. Would you be willing to provide an example of a "Complex" kernel? Would parallelizing pixel sorting be considered complex because of how merging the threads' results back together works? Is there a better way we can get more speedup than just making our functions more complicated?

General Progress

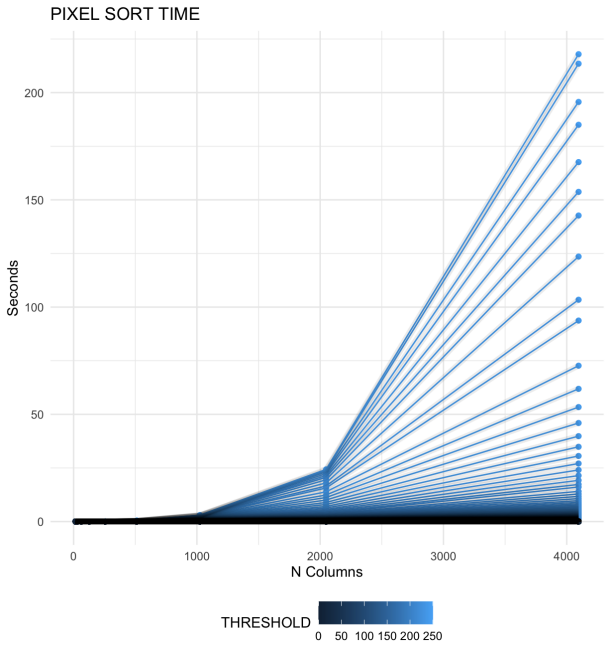
Our project is off to a wonderful start. We have been able to implement a serial version of every single one of our filters. Our project design stated that we hoped to have serial versions of everything except for background removal, but we were able to complete all serial code. Additionally, we have made significant progress on our kernels. Greyscale, rotation, and background removal, all have kernels implemented. This is also significantly ahead of where we hoped to be at this point in the semester. In addition to the things detailed in the project specifications, we have also made progress towards a video demo of some of our functions. This is a great way of showing exactly what pixel sorting is, and how it works. We have also experimented with a GUI on the front end to expand our domain of users. This will make our software more paletable to people unfamiliar with the command line.

By the mid-project draft we expected to have the serial code written for every function except the background removal. Thanks to the diligence our team we managed to get that working too. We also expected to have the desaturation, Gaussian blur, and rotate kernels implemented. We also planned on having a testing suite for desaturation, Gaussian blur, and rotation which will validate both our serial and parallelized implementations. We were able to complete all of these tasks and more. We are well on our way to finishing the project, and will hopefully have time to add extra components not described in our initial project design.



Updates

The one place our project is currently lacking is the implementation of OpenCV. Since this was a non-essential functionality we decided to just scrap it all together so that we could focus more of our efforts on the core functionality of our project. There are a few things we need to finish before we are ready to submit the project. The main two things are that we need to finish our kernels, and complete some more extensive testing. One form of testing we might implement is comparing the serial and parallel versions of output images. The images should be exactly the same for both programs when the same filters are run on them. We also need to do some more performance review for our final paper. Additionally, we hope to finalize and add more videos of our other functions. Finally, if time allows, we would like to implement this GUI front end.

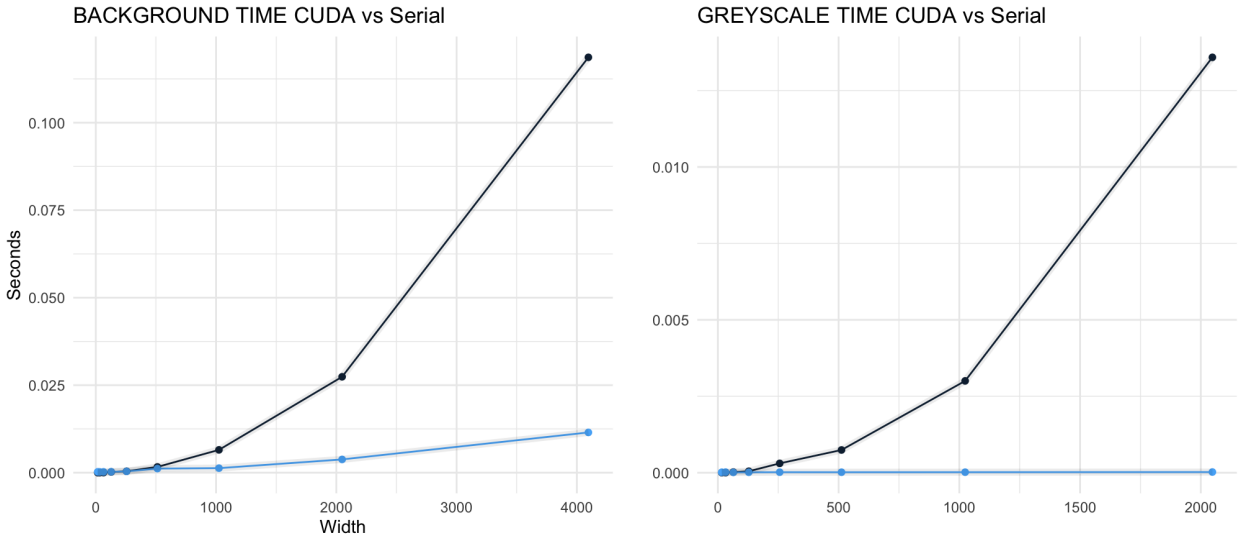


When setting up our serial image processing C program, there were several roadblocks that we encountered. Initially, we tried to use netpbm.h, which we had used successfully in the CUDA lab. However, we made the assumption that since it was working on PNG images, it would handle transparent pixels. Unfortunately, we were wrong, and had to go back to the drawing board after our initial serial implementation was complete. After conducting some more research surrounding the issue, we decided that libpng was the best alternative. It was a very tedious process to refactor the existing code to use libpng since we had to scrap our original read/write utility, but we were able to find some example read/write functions for libpng that helped us get things running again. This process was time-consuming but necessary since we planned on using the background removal functions to apply masks to the pixel sorting feature.

Taking an existing serial function and converting it to a working CUDA kernel was, unsurprisingly, a huge obstacle. Careful consideration of memory allocation was needed to move data correctly to GPU memory in order to perform GPU computations. Researching how libpng allocated memory for its read function was crucial to figuring out a CUDA implementation. Tinkering with the original libpng read function proved to be too difficult, so we compromised with a basic copying operation into new CUDA synced memory. Coupled with trial an error, we successfully transferred the necessary png image information onto synced CUDA memory. Afterwards, modifying the serial imaging processing functions to account for the parallel nature of GPU computation was a chore. Establishing the technique of distributing work across grid and threads is conceptually tough. Nonetheless, we built working solutions for some of our basic filters. The next major roadblock that we encountered was that we experienced very slow saving speed. It slowed down our testing procedures to a crawl. Upon further investigation, we discovered that the issue was in the write utility. After reviewing documentation of libpng, we discovered that a medium amount of image compression was enabled by default in the write function. For the time being we have disabled this compression, which immensely improved the write speed of massive input files. Testing will be far faster as a result.

Final Project Deliverable

Finishing our final project will entail refining what we have and (possibly) creating a better front-end. For example, there are an excessive amount of background removal functions with overly simplistic formulas. Consolidating them and adding more complex computations will not only produce more accurate results but take better advantage of the GPU. Our current, basic background removal is only providing a 5x speedup in comparison to serial performance. We plan on refining the current process to a new one with CIELUV color thresholding, distance metrics, median filters, and more. A more computationally expensive function should (hopefully) provide more contrast between CPU serial performance and GPU CUDA accelerated performance. We plan to submit our code as well as some example images to use. This code will use at least 5 kernels, one for each image processing technique. As shown in class, image processing techniques benefit greatly from parallelization because of the amount of data being performed on the same task. We expect all of our kernels to demonstrate an over 10x speedup, and we should have no trouble getting over 20x speed up. The pixel sorting algorithms will require more thought when parallelizing because each thread needs to know where the other threads are placing their sorted pixels to avoid a data race. This code will also demonstrate a "one-touch" testing framework and a fully-correct write-up detailing the analysis of every kernel. Lastly we plan on including a section detailing future work. Because we have included everything mentioned above this project demonstrates an "A" level of work.



CUDA shown in light blue.

Acknowledgements

Special thanks to Dr. Lam for introducing us to CUDA and its benefits. Learning about this powerful tool has been an eye-opening experience that has greatly enhanced our understanding of parallel programming. His guidance and expertise has been instrumental in helping us understand the intricacies of this technology. Through his clear and concise explanations, we have gained a deep appreciation for the efficiency and scalability that OpenMP can provide. We are thrilled to continue exploring the possibilities of OpenMP and CUDA then applying this knowledge to our future projects.

References

- `https://github.com/DavidMcLaughlin208/PixelSorting`
- `https://developer.nvidia.com/blog/even-easier-introduction-cuda/`