15-418 Project Proposal

CUDA Program Debugger/Optimizer

URL: https://kflorendo.github.io/15418-final-project/

SUMMARY

Our project is a CUDA program debugger/optimizer. The debugger helps analyze the results of incorrect code and determine potential areas of error. For example, it will be able to check the correctness of output for each thread, and point out issues such as overwriting shared memory. This tool would help users debug their CUDA code more efficiently, without needing to add streams of print statements. Further, the optimizer helps users tune parameters of their program, like CUDA grid configurations, to determine what will achieve the best performance. The optimizer also calculates time and memory metrics to identify bottlenecks and understand overall program performance.

BACKGROUND

We plan to implement a CUDA program debugging/optimizing tool to help students and developers understand and visualize the behavior of their program.

Primary Feature 1: Debugging

The output of each individual thread will be written to a file so they can be analyzed later. Additionally, the program will check for overwriting by tracking the number of times a variable has been modified before it is accessed. For example, if a thread returns an array of RGB values, but the rendered image values do not match, asking the debugger to track the variable would print out the value of the variable each time it was accessed, as well as the thread that made each access. If time permitting, we would like the debugger to also suggest a possible source of the problem, such as "variable was accessed by thread 2, 64. Please verify your chunk_size calculations".

Primary Feature 2: Optimizing

We will write scripts that will run the program multiple times with different CUDA grid configurations and output the performance of each to help the programmer decide on the optimal configuration. This can be done using bash scripts that modify the program files. First, we will tell users how to initialize these parameters to allow for modification by our application. For instance, "Please define the number of threads per block at the top of your .cu file as follows: `#define THREADS_PER_BLOCK 1024`". Then, the user will enter the command they wish to run (i.e. `./render rgb`). Our script will then look for these macros and modify them to test out different parameters. For different configurations of these macros, we'll run the command the user entered. Our script can also insert timing code into the user's program to calculate metrics and make comparisons across configurations. If time permits, we also hope to optimize running these different program versions in parallel.

Another feature we want to add is analyzing sections of the code to determine time and memory bottlenecks. Using bash scripts, we can insert timing code around user-specified sections of the code (kernels, groups of kernels, sequential code). We can also insert timing code around memory access operations to help users determine what kind of memory they should use (global, shared, or thread-local).

THE CHALLENGE

Something we may need to attempt is parallelizing running the parallel code, since optimization would require running the same code several times with adjusted parameters. We hope to have a better conceptual grasp of what may cause errors in CUDA, and what patterns there may be in optimizing code. A potential concern is that the efficiency of the CUDA debugger may largely depend on the efficiency of the code it is running on, so a feature that attempts to optimize code without necessarily running it would be helpful. Essentially, we are using the knowledge we learned in this class about CUDA as well as our past struggles to try to make coding and debugging in CUDA more straightforward.

RESOURCES

We will be using our past assignments as test code for our debugger. In terms of code, we will mostly be starting from scratch, apart from some starter code to create a basic GUI. We will be referencing resources about debugging/optimizing, like the lecture on Performance Measurement and Tuning, or other articles we find online. For GUI, we will be referencing the documentation (www.gtk.org/docs/) for starter code and setup. Certain features of our debugger may require running the code several times with a few CUDA command changes (ex. Number of threads), so access to PSC machines would ensure that the debugger would terminate faster.

GOALS AND DELIVERABLES

Plan To Achieve

- Program is able to write the output of all threads to a file, along with thread number indications. We should be able to achieve this goal because we have enough experience with CUDA to be able pinpoint what needs to be done.
- Given a variable or array, the program is able to detect overwriting, if any exists. We should be able to achieve this goal because during our previous CUDA assignment, we had to address the same issue by adding several print statements. Because of this experience, we are conceptually familiar with how this can be implemented.
- Program is able to write scripts that run the given program multiple times with different CUDA grid configurations. The performance of each configuration will be outputted so the user can determine the optimal configuration.

- Program can detect time and memory bottlenecks by adding timer code. We should be able to achieve this goal because we have had experience with timing code to see which part is influencing performance the most.

Hope To Achieve

- Implement feature that suggests a possible source of the overwriting. (ex. chunk_size)
- Parallelize running the different CUDA configurations, which will speed up the performance of the debugger.
- Implement feature that suggests to the user what type (global, shared, thread-local) would be best given a variable/array
- Generate graphs from the results of running different CUDA grid configurations

At the poster session, we hope to have an interactive demo of running the debugger on a CUDA program (ex. Assignment 2). The program should be able to detect the errors listed above, and hopefully generate graphs and statistics that help the user refine certain variables and parameters. Through this project, we hope to more thoroughly understand CUDA, as well as the common errors that can stem from it. We also hope for the tool to be helpful to those who may use CUDA in the future by saving time for debugging and trying similar implementations.

PLATFORM CHOICE

We will be using bash for scripting, which works well for reading/writing files and executing commands in the terminal. We'll use C++ for creating the GUI, particularly the GTK library and gtkmm, which is the C++ interface for the library. We chose C++ compared to Python and other languages for its speed. Finally, for the code to calculate metrics and analyze program performance, we will be using CUDA and C/C++.

SCHEDULE

Date	Goals and Deadlines
4/3 ~ 4/9	Learn GTK basics and start basic GUI (all screen layouts and components to take in input/display output). If possible, implement task for writing the output of all threads to a file, with thread number indications.
4/10 ~ 4/16	Detection for thread overwriting. Given a particular variable or array, the program will save the value(s) stored each time it is accessed, as well as the thread number of the thread that accessed it. Finish basic GUI from last week.
	If time permitting, in addition to overwriting detection, implement feature that suggests possible source of overwriting
4/17 ~ 4/19	Add finishing touches to debugging section, work on milestone report, and transition to optimizing portion of project
4/19	Milestone Report Due
4/20 ~ 4/23	Write scripts that runs program multiple times with different CUDA grid configurations, and output the performance of each to help determine optimal configuration.
	If time permitting, implement program so the above programs with different configurations will run in parallel, and generate graphs of metrics such as memory accesses over time to help compare performance
4/24 ~ 4/30	Implement code that detects time and memory bottlenecks. Refine GUI.
	If time permitting, add feature to suggest the kind of memory user should implement (global, shared, or thread-local)
5/1 ~ 5/3	Add finishing touches, work on final report
5/4	Final Report Due