15-418 Milestone Report

CUDA Program Debugger/Optimizer

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

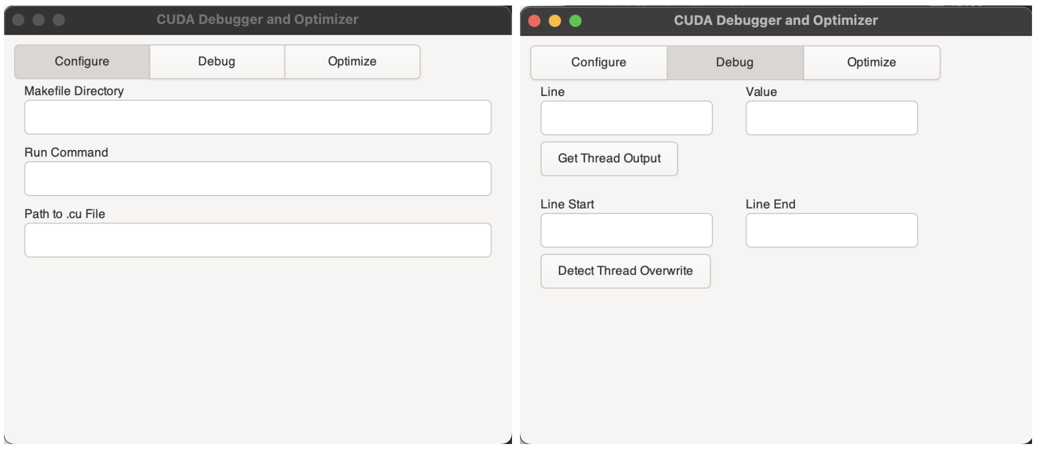
I. Updated Schedule

Date	Goals and Deadlines	Name
4/3 ~ 4/6	Learn GTK basics and start basic GUI (all screen layouts and components to take in input/display output). Begin implementing task for writing the output of all threads for a specific variable to a file, with thread indices.	Katrina
4/3 ~ 4/6	Learn basics of bash scripts, plan outline of how to implement thread overwriting.	Huining
4/6 ~ 4/9	Finished implementing task for writing the output of all threads for a specific variable to a file, with thread indices.	Katrina
4/6 ~ 4/9	Detection for thread overwriting. Given a particular variable, the program will save the value(s) indicating which thread made the change each time the variable is accessed.	Huining
4/10 ~ 4/13	Finish basic GUI from last week.	Katrina
4/10 ~ 4/13	In addition to overwriting detection, implement a feature that suggests a possible source of overwriting. For the same address that is written to multiple times by different threads, the corresponding address will be printed along with the blockIdx and threadIdx of all the threads that made changes.	Huining
	CARNIVAL WEEKEND	
4/17 ~ 4/19	Add finishing touches to debugging section, work on milestone report	Katrina, Huining
4/19	Milestone Report Due	
4/20 ~ 4/23	Implement code that detects time bottlenecks by writing a script that places timers in respective areas, input is list of line number tuples.	Huining
4/20 ~ 4/23	Write scripts that runs the program multiple times with different CUDA grid configurations, and output the performance of each to help determine optimal configuration.	Katrina
	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	Katrina. Huining

4/24 ~ 4/26	Plan out how to and begin implementing code that detects memory bottlenecks. Refine GUI. Work on above time permitting task if possible.	Katrina. Huining
	~~If time permitting, add feature to suggest the kind of memory user should implement (global, shared, or thread-local)~~	
4/27 ~ 4/30	Implement code that detects memory bottlenecks, refine GUI, connect GUI to bash scripts, and test on Windows machine.	Katrina, Huining
5/1 ~ 5/3	Add finishing touches, work on final report	Huining, Katrina
5/4	Final Report Due	

II. Work Completed Thus Far

So far, we have completed all tasks that were planned in the schedule with only some minor changes being made. We have a basic GUI completed with minimalistic screen layouts and all necessary components included. The bash scripts for writing the output of threads as well as their corresponding indices responsible for changing a given variable to a file works as intended, and similarly, the bash script for detecting thread overwriting is completed. Both scripts currently can be used on singular variables and arrays. In the upcoming few days, we plan on adding an option of using the above functions on 2-dimensional grids.



The GUI is built with C++, gtkmm, and glade. It consists of 3 screens: Configure, Debug, and Optimize. In the Configure tab, users are asked to input configuration details for their CUDA program, including the directory containing the Makefile, the command to run the executable, and the path to the .cu file. In the Debug tab, there are buttons for running the thread output and overwriting scripts, and text entry boxes for passing arguments to these scripts. The bash script for getting thread outputs works by inserting code into the CUDA file that saves the blockIdx, threadIdx, and variable value to a text file, compiling the code, and running the executable. The script takes in as arguments the variable and location in the code file to observe. On the other hand, the bash script for detecting overwrites writes the blockIdx, threadIdx, and the address of the changed variable into a text file. Then, the addresses are used as keys in a dictionary-like structure called associative arrays, with the blockIdx and threadIdx as a tupled value. If a key corresponds to multiple values, then the system will print out all the tupled values assigned to that address, since the thread that is responsible for the overwriting must be one of the options listed in the dictionary values.

III. Goals and Deliverables Update

We are relatively on schedule with respect to the goals and deliverables stated in our proposal. We believe that we will still be able to produce all our deliverables with minor changes only if necessary. One of the stretch goals we had for the proposal was implemented, while one of our later stretch goals of adding a feature that suggests the kind of memory the user should implement was removed due to time constraints, however, if possible, we do plan on attempting to implement this.

Goals: (crossed out if completed, highlighted are stretch goals)

- ~~Implement basic GUI~~
- ~~Write bash script for writing the output of all threads for a specific variable to a file~~
- ~~Write bash script to detect thread overwriting, and output possible sources (thread indices)~~
- Write script that detects time bottlenecks in code
- Write script that runs program multiple times with different CUDA grid configurations
- Implement program that outputs graphs representing the above configuration data
- Implement program that runs the program-running code in parallel
- Write script that detects memory bottlenecks in code
- Connect GUI to bash scripts, test, and refine GUI

IV. Poster Session Details

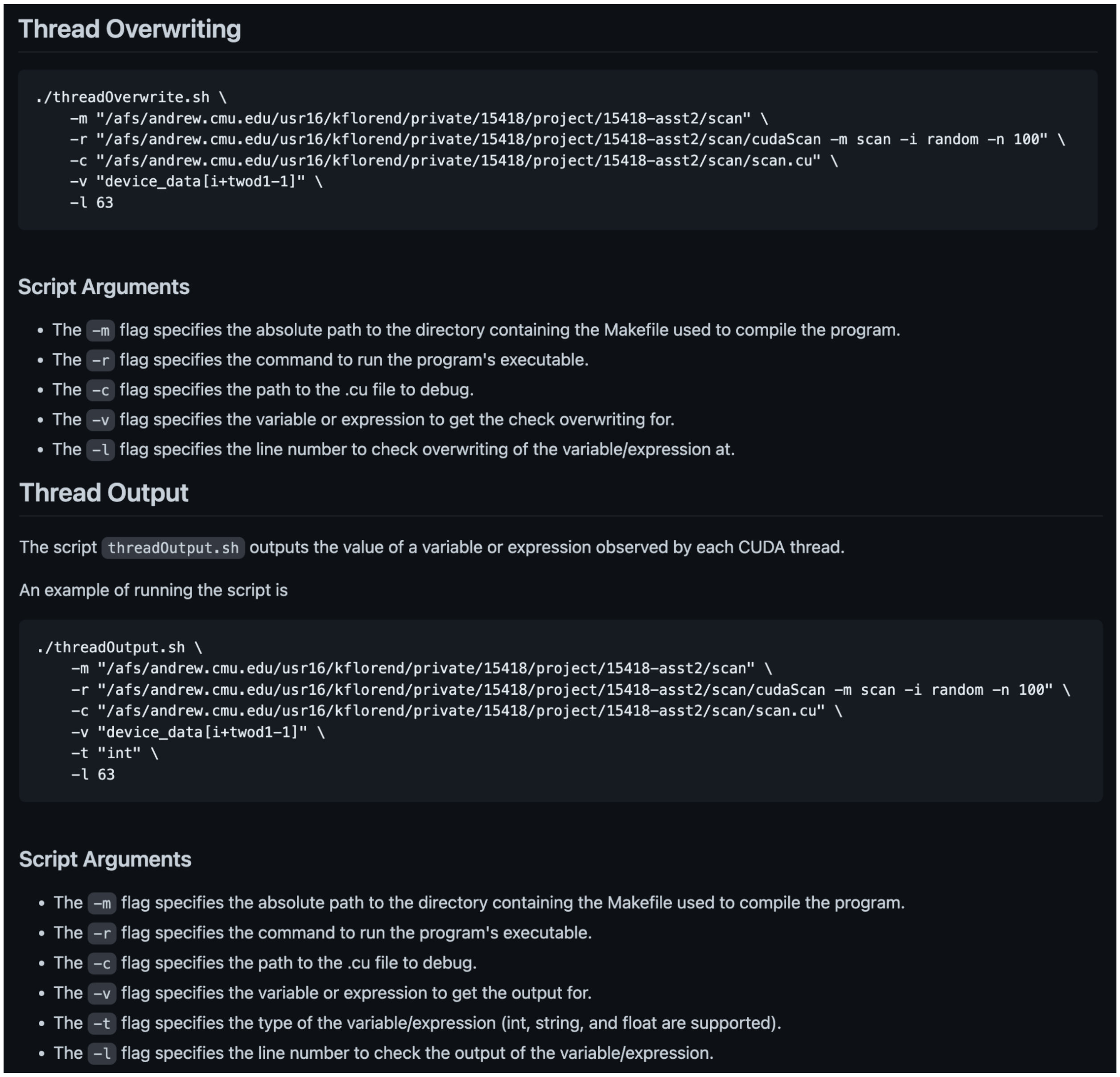
We hope to display both a demo and the graphs generated by the demo. The graphs would depict the data showing which configuration is optimal for the given code, while the demo would allow user interaction and display all other functions of our debugger.

V. Preliminary Results

GitHub Links

GitHub link for bash scripts: https://github.com/kflorendo/cuda-debugger-optimizer-scripts
GitHub link for GUI: https://github.com/kflorendo/cuda-debugger-optimizer

Running the Bash Scripts



Thread Overwriting

```
./threadOverwrite.sh \
    -m "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan" \
    -r "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan/cudaScan -m scan -i random -n 100" \
    -c "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan/scan.cu" \
    -v "device_data[i+twod1-1]" \
    -l 63
```

Script Arguments

- The `-m` flag specifies the absolute path to the directory containing the Makefile used to compile the program.
- The `-r` flag specifies the command to run the program's executable.
- The `-c` flag specifies the path to the .cu file to debug.
- The `-v` flag specifies the variable or expression to get the check overwriting for.
- The `-l` flag specifies the line number to check overwriting of the variable/expression at.

Thread Output

The script `threadOutput.sh` outputs the value of a variable or expression observed by each CUDA thread.

An example of running the script is

```
./threadOutput.sh \
    -m "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan" \
    -r "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan/cudaScan -m scan -i random -n 100" \
    -c "/afs/andrew.cmu.edu/usr16/kflorend/private/15418/project/15418-asst2/scan/scan.cu" \
    -v "device_data[i+twod1-1]" \
    -t "int" \
    -l 63
```

Script Arguments

- The `-m` flag specifies the absolute path to the directory containing the Makefile used to compile the program.
- The `-r` flag specifies the command to run the program's executable.
- The `-c` flag specifies the path to the .cu file to debug.
- The `-v` flag specifies the variable or expression to get the output for.
- The `-t` flag specifies the type of the variable/expression (int, string, and float are supported).
- The `-l` flag specifies the line number to check the output of the variable/expression.

Bash Script Outputs

We tested our implementations on the 15418-Assignment 2 CUDA code for scan.

```
threadIdx.x blockIdx.x device_data[i+twod1-1]
3 0 23
3 1 16
1 0 2
1 1 4
1 2 19
1 3 7
1 4 23
...
```

Excerpt from the output text file for threadOutput.sh

The output text file for threadOutput.sh contains the columns threadIdx, blockIdx, and the values of the variable that is being examined. The threadId can be calculated using the first two numbers.

```
address 2537594876 written to by (0, 79),(0, 78),(0, 77)
address 2537566844 written to by (0, 100),(0, 96),(0, 92)
address 2537564156 written to by (0, 79),(0, 39),(0, 19),(0, 75),(0, 37),(0,
18),(0, 71),(0, 35),(0, 17)
address 2537560700 written to by (0, 105),(0, 52),(0, 97),(0, 48),(0, 89),(0,
44)
address 2537555172 written to by (0, 92),(0, 28)
address 2537557628 written to by (0, 115),(0, 57),(0, 28),(0, 99),(0, 49),(0,
24),(0, 83),(0, 41),(0, 20)
address 2537556076 written to by (0, 102),(0, 70)
address 2537560508 written to by (0, 102),(0, 94),(0, 86)
...
```

Excerpt from the output text file for threadOverwrite.sh

The final output text file for threadOverwrite.sh returns the address that is changed, as well as a list of tuples containing (blockIdx, threadIdx) corresponding to the address.

VI. Concerns/Issues

- Time Constraints - we are unable to make a completely accurate estimation on how long each task will take, and we are also unsure of the amount of time needed to complete our other responsibilities.
- How to implement memory bottleneck detection - Initially, this idea emerged alongside time bottleneck detection, but we realized that the implementations between the two would be quite different, since the latter is more straightforward to implement (via placing timers throughout the code), but the same cannot be said for memory.
- How to implement a feature suggesting what kind of memory the user should implement - hypothetically, this could be done by replacing the variable type with different options and measuring the performance of each with our other implemented functions. However, we are unsure if there is a more robust way of achieving this.
- How to generate graphs - we have the option of generating graphs via C++ and displaying it in the GUI, or generating the graphs externally using the data points that will be outputted into a text file.