Verification of the Prefix Sum Program in an OpenCL Environment

Thijs Wiefferink
thijs@wiefferink.me, s1366564

University of Twente

Abstract

todo: write

Keywords

todo: write

Preface

This project is the continuation of my bachelor thesis project, in which the verification of the prefix sum algorithm has been started. Since only the 'data race free' part has been proven for the first part of the algorithm (the upsweep), the goal of this project is to prove the complete algorithm data race free, and additionally prove the functionality of the algorithm.

The necessary background information will be included in this report to understand the goal and results of the project, but full details of the Bachelor project can be read in the paper of that project [3].

Contents

1 Introduction

This chapter describes the research domain, shows the problem that is solved, introduces the research questions and explains the approach.

1.1 Research domain

In this section the context information of this research project is described.

1.1.1 GPU computing

A graphics processing unit (GPU) is a device designed to rapidly manipulate and alter memory to accelerate the creation of images, for example while watching a video or playing a game. However, GPUs are also used more for general purpose computing, which is traditionally handled by the central processing unit (CPU). GPUs are better than CPUs doing parallel execution on large data sets. For example increasing the brightness of an image is easily done by a GPU, since this operation can be done in parallel on all pixels of the image. GPUs are however also used for physics calculations or mining crypto currencies. When using a GPU for general computing an API has to be used, I have chosen OpenCL for the previous research project because of its hardware vendor independency and open source nature.

Running a parallel computation on a GPU brings a couple of challenges. The first challenge is preventing data races. A data race is the situation in a program where multiple threads are accessing the same memory location, with at least one of them writing to the location. The second challenge is verifying the correctness of the functionality of the program. Verifying both of these aspects is useful for safety critical systems.

The previous research project has started with the verification process to show that a program (specifically the Prefix Sum algorithm) has no data races. Since only the first half of the program could be verified in the given time for the project the verification has been continued in this project.

1.1.2 Prefix Sum

To show what the prefix sum is and how it is calculated, this section repeats the information from the previous project[3] below.

The algorithm computes the sums of all possible prefixes of an input array. In Figure 1, a mathematical representation of the prefix sum is illustrated, x represents the input array, x_0 indicates the first element from the input array, where n represents the size of the input array, and y is the output array containing the prefix sums. Each number y_a in the output array is the sum of all numbers $x_b \in x$ for which the condition $b < a$ holds.

The Prefix Sum algorithm is an interesting case study because it is a building block for a lot of other algorithms. For example radix sort and quicksort can be implemented using Prefix Sums, but it can also be used to lexically compare strings of characters or to search for regular expressions [1]. In the field of specifying and verifying GPU programs the Prefix Sum is a suitable next step, because it will be a bigger and more complex example of verifying a GPU program.

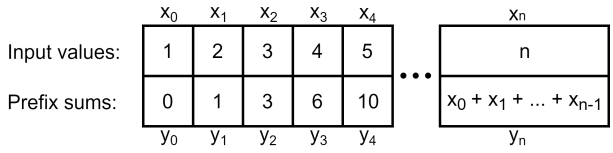

Figure 1: Prefix Sum description

The algorithm to calculate prefix sums can be structured in such a way that large amounts of data can be processed in parallel. A multi threaded algorithm meant for the GPU has been implemented in the previous project. The verification process continues with this same algorithm. The implemented version of the Prefix Sum is based on Chapter 39 *Parallel Prefix Sum (Scan) with CUDA* of the book GPU Gems 3 [2].

1.1.3 Verification

For the verification of the Prefix Sum program Permission-Based Separation Logic is used to specify the behavior of the program, and the tool VerCors is used to verify that the code matches the specification. A description of Permission-Based Separation Logic can be found in the previous project[3].

1.2 Problem statement

Making sure a program has no data races, and always gives correct results is extremely hard, if not impossible with traditional testing. For a single-threaded application testing all inputs should prove that the program is correct, but for multi-threaded applications this does not prove anything about data races. In order to verify the correctness of a program it has to be specified in Permission-Based Separation Logic, which is a challenge for bigger programs like the Prefix Sum. VerCors, the tool used to verify the specification, will get slower when a larger specification is used. This makes using the tool correctly and efficiently a challenge. During the previous project a number of problems in the tools have been identified that blocked the verification, during this project this will be the case as well.

1.3 Research questions

The following research questions have been formed from the problem statement:

1. How can the Prefix Sum program be proven to have no data races?
2. How can the Prefix Sum program be proven to give the correct result?
3. What are the limitations of VerCors for verifying GPU programs?

The first research question has partially been proven by the previous research, which will be used as a starting point. To answer it fully the specification of the Prefix Sum algorithm has to be extended to the complete program (add the downsweep and final permissions). For the second research question the specification created for the first question has to be extended with information about the results. The third question is to review the VerCors tool, which will be done during the verification process of the first two questions.

1.4 Approach

First the last specification of the previous project will be tested in the last version of VerCors, to ensure it still works after all changes in tool. After that works, the verification of the read/write permissions can be continued for the downsweep phase of the program. I expect that the specification should not be hard, because it uses much of the same patterns of the upsweep phase, but getting VerCors to actually verify it will take time. After the downsweep phase is specified and verified, the `ensures` clause of the complete program can be added, which might give some trouble to proof as well. While adding more specifications I expect the verification time to increase. Hopefully this does rise to a time that makes iterating through different versions of the specification too slow.

After the read/write permissions have been verified, the functional specification and verification can be started. To speed up this process it is probably best to remove the downsweep phase at first, and just start with the upsweep phase. This will keep the iteration time low and should make it easy to rapidly expand the specification. Doing the verification of the downsweep phase might get slow due to the verification time of VerCors. Adding the final ensures for the complete program might be the hardest task, since the everything needs to be enabled at that point.

To answer the third research question notes will be made during the verification process to keep track of tool improvements and limitations. These will be written down to provide future work for the tool, or notes for creating specifications that are supported by VerCors.

When encountering a problem during the verification, Stefan Blom (author of VerCors) will be contacted to see if there is a problem in VerCors and if/how that could be solved.

1.5 Report structure

todo: introduce chapters

2 Prefix sum algorithm

todo: explain parallel prefix sum, based on bachelorreferaat paper

3 Verifying permissions

todo: verification process of the read/write permissions of the array (initial based on bachelorreferaat, extended to allow functional verification)

4 Verifying functionality

todo: verification process of the functionality of the program

5 Discussion

todo: summary

5.1 Results

todo: result description: verified

5.2 Related work

todo: similar verification projects

6 Conclusion

todo: summary

6.1 Research question answer

todo: works, not for huge programs though

6.2 Limitations and problems

todo: tool limitations, change code for verification process

6.3 Research value

todo: do larger programs, improve tool, concurrency

6.4 Future work

1. Larger programs
2. Improve tool
3. Handle more functional details: overflow, underflow, etc.

7 References

[1] G. E. Blelloch. Prefix sums and their applications. 1990.

[2] H. Nguyen. *GPU Gems 3*. Addison-Wesley Professional, first edition, 2007.

[3] T. Wiefferink. Optimization, specification and verification of the prefix sum program in an opencl environment. 2015.