Interpreter and Transpiler for simple expressions on Nvidia GPUs using Julia

Daniel Wiplinger

MASTERARBEIT

eingereicht am

Fachhochschul-Masterstudiengang

Software Engineering

in Hagenberg

im Januar 2025

Advisor:

DI Dr. Gabriel Kronberger

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, January 1, 2025

Daniel Wiplinger

Contents

Abstract

This should be a 1-page (maximum) summary of your work in English.

Kurzfassung

An dieser Stelle steht eine Zusammenfassung der Arbeit, Umfang max. 1 Seite. ...

Chapter 1

Introduction

This chapter provides an entry point for this thesis. First the motivation of exploring this topic is presented. In addition, the research questions of this thesis are outlined. Lastly the methodology on how to answer these questions will be explained.

1.1 Background and Motivation

Optimisation and acceleration of program code is a crucial part in many different fields. For example video games need optimisation to lower the minimum hardware requirements which allows more people to run the game. Another example where optimisation is important are computer simulations. For those, optimisation is even more crucial, as this allows the scientists to run more detailed simulations or get the simulation results faster. Equation learning is another field that can heavily benefit from optimisation. One part of equation learning, is to evaluate the expressions generated by the algorithm which can make up a significant portion of the runtime of the algorithm. This thesis is concerned with optimising the evaluation part to increase the overall performance of the equation learning algorithm.

Considering the following expression $x_1 + 5 - \text{abs}(p_1) * \text{sqrt}(x_2)/10 + 2\hat{}3$ which contains simple mathematical operations as well as variables x_n and parameters p_n. This expression is one example that can be generated by the equation learning algorithm and needs to be evaluated for the next iteration. Usually multiple expressions are generated per iteration, which also need to be evaluated. Multiple different values need to be inserted for all variables and parameters, drastically increasing the amount of evaluations that need to be performed.

The free lunch theorem as described by Adam et al. (2019) states that to gain additional performance, a developer cannot just hope for future hardware to be faster, especially on a single core. Therefore, algorithms need to utilise the other cores on a processor to further acceleration. While this approach means more development overhead, a much greater speed-up can be achieved. However, in some cases the speed-up achieved by this is still not large enough and another approach is needed. One of these approaches is the utilisation of a Graphics Processing Unit (GPU) as an easy and affordable option as compared to compute clusters. Michalakes and Vachharajani (2008) have shown a noticeable speed-up when using the GPU for weather simulation. In addition to computer simulations GPU acceleration also can be found in other places like networking (Han et al., 2010) or structural analysis of buildings (Georgescu et al., 2013).

1.2 Research Question

With these successful implementations of GPU acceleration, this thesis also attempts to improve the performance of evaluating mathematical equations using GPUs. Therefore, the following research questions are formulated:

- How can simple arithmetic expressions that are generated at runtime be efficiently evaluated on graphics cards?
- Under what circumstances is the evaluation of simple arithmetic expressions faster on a graphics card than on a CPU?
- Under which circumstances is the interpretation of the expressions on the GPU or the translation to the intermediate language Parallel Thread Execution (PTX) more efficient?

In order to answer these questions, two GPU expression evaluators need to be implemented. The first evaluator will interpret the expressions entirely on the GPU, while the second will transpile them to PTX-Code on the CPU and execute the generated code on the GPU. Research needs to be done to explore different possibilities to implement the two evaluators. The current implementation of the equation learning algorithm already contains a CPU expression evaluator, which will be used to compare the GPU evaluators against.

1.3 Methodology

Will give an overview of the chapters and what to expect

Chapter 2

Fundamentals and Related Work

2.1 Equation learning

Section describing what equation learning is and why it is relevant for the thesis

2.2 General Purpose Computation on Graphics Processing Units

Describe what GPGPU is and how it differs from classical programming. talk about architecture (SIMD) and some scientific papers on how they use GPUs to accelerate tasks

2.2.1 Parallel Thread Execution

Describe what PTX is to get a common ground for the implementation chapter. Probably a short section

2.3 GPU Interpretation

Different sources on how to do interpretation on the gpu (and maybe interpretation in general too?)

2.4 Transpiler

talk about what transpilers are and how to implement them. If possible also gpu specific transpilation. Also talk about compilation and register management. and probably find a better title

Chapter 3

Concept and Design

introduction to what needs to be done. also clarify terms "Host" and "Device" here

3.1 Requirements and Data

short section. Multiple expressions; vars for all expressions; params unique to expression; operators that need to be supported

3.2 Interpreter

as introduction to this section talk about what "interpreter" means in this context. so "gpu parses expr and calculates"

3.2.1 Architecture

talk about the coarse grained architecture on how the interpreter will work. (.5 to 1 page probably)

3.2.2 Host

talk about the steps taken to prepare for GPU interpretation

3.2.3 Device

talk about how the actual interpreter will be implemented

3.3 Transpiler

as introduction to this section talk about what "transpiler" means in this context. so "cpu takes expressions and generates ptx for gpu execution"

3.3.1 Architecture

talk about the coarse grained architecture on how the transpiler will work. (.5 to 1 page probably)

3.3.2 Host

talk about how the transpiler is implemented

3.3.3 Device

talk about what the GPU does. short section since the gpu does not do much

Chapter 4

Implementation

4.1 Technologies

Short section; CUDA, PTX, Julia, CUDA.jl

Probably reference the performance evaluation papers for Julia and CUDA.jl

4.2 Interpreter

Talk about how the interpreter has been developed.

4.3 Transpiler

Talk about how the transpiler has been developed

Chapter 5

Evaluation

5.1 Test environment

Explain the hardware used, as well as the actual data (how many expressions, variables etc.)

5.2 Results

talk about what we will see now (results only for interpreter, then transpiler and then compared with each other and a CPU interpreter)

5.2.1 Interpreter

Results only for Interpreter

5.2.2 Transpiler

Results only for Transpiler

5.2.3 Comparison

Comparison of Interpreter and Transpiler as well as Comparing the two with CPU interpreter

Chapter 6

Conclusion and Future Work

Summarise the results

6.1 Future Work

talk about what can be improved

References

Literature

Adam, S. P., Alexandropoulos, S.-A. N., Pardalos, P. M., & Vrahatis, M. N. (2019). No free lunch theorem: A review. In I. C. Demetriou & P. M. Pardalos (Eds.), *Approximation and optimization : Algorithms, complexity and applications* (pp. 57–82). Springer International Publishing. https://doi.org/10.1007/978-3-030-12767-1_5. (Cit. on p. 1)

Georgescu, S., Chow, P., & Okuda, H. (2013). GPU acceleration for FEM-based structural analysis. *Archives of Computational Methods in Engineering*, *20*(2), 111–121. https://doi.org/10.1007/s11831-013-9082-8 (cit. on p. 2)

Han, S., Jang, K., Park, K., & Moon, S. (2010). PacketShader: A GPU-accelerated software router. *SIGCOMM Comput. Commun. Rev.*, *40*(4), 195–206. https://doi.org/10.1145/1851275.1851207 (cit. on p. 2)

Michalakes, J., & Vachharajani, M. (2008). GPU acceleration of numerical weather prediction [ISSN: 1530-2075]. *2008 IEEE International Symposium on Parallel and Distributed Processing*, 1–7. https://doi.org/10.1109/IPDPS.2008.4536351 (cit. on p. 1)

Check Final Print Size

— Check final print size! —

width = 100mm
height = 50mm

— Remove this page after printing! —