Masaryk University
Faculty of Informatics

Framework for Parallel Kernels Autotuning

Master's Thesis

Bc. Filip Petrovič

Brno, Fall 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Filip Petrovič

Advisor: RNDr. Jiří Filipovič, Ph.D.

Acknowledgements

I would like to thank my supervisor Jiří Filipovič for his help and valuable advice. I would also like to thank my family for their support during my work on the thesis.

Abstract

The result of this thesis is a framework for autotuning of parallel kernels which are written in either OpenCL or CUDA language. The framework includes advanced functionality such as support for composite kernels and online autotuning. The thesis describes API and internal structure of the framework and presents several examples of its utilization for kernel optimization.

Keywords

autotuning, GPU programming, OpenCL, CUDA, kernel, optimization

Contents

1 Introduction

In recent years, acceleration of complex computations using multi-core processors, graphics cards and other massively parallel devices has become much more common. Currently, there are many devices developed by multiple vendors which differ in hardware architecture, performance and other attributes. In order to ensure portability of code written for particular device, several software APIs (application programming interfaces) such as OpenCL (Open Computing Language) or CUDA (Compute Unified Device Architecture) were designed. Code written in these APIs can be run on various devices, while always producing the same result. However, there is a problem with portability of performance. For example, code which was optimized for a GPU may run poorly on a regular multi-core processor. The problem also exists among multiple generations of devices developed by the same vendor, even if they have comparable parameters and theoretical performance.

A costly solution to this problem is to manually optimize code for each utilized device. This has several significant disadvantages, such as a necessity to dedicate large amount of resources to write different versions of code and test which one performs best on a given device. Furthermore, new devices are released frequently and in order to efficiently utilize their capabilities, it is often necessary to rewrite old versions of code and repeat the optimization process again.

An alternative solution is a technique called autotuning, where code includes parameters which affect performance depending on their value, for example a parameter which affects length of a vector type of a particular variable. Optimal values of these parameters might differ for various devices based on their hardware capabilities. Parametrized code is then launched repeatedly using different combinations of parameters to find out the best configuration for a particular device.

In order to make autotuning easier to implement in applications, several frameworks were created. However, large number of these are focused only on a very small subset of computations. There are some frameworks which are more general, but their features are limited and only support simple usage scenarios. The aim of this thesis was to

develop autotuning framework which would support more complex use cases, such as situations where computation is split into several smaller programs. Additionally, the framework should be written in a way which would allow its easy integration into existing software stacks and possibly combine autotuning with regular computation.

The thesis is split into four main chapters. <Todo: add short description of each chapter.>

2 Compute APIs and autotuning

This chapter serves as an introduction to autotuning technique and includes description of compute APIs which are utilized by KTT (Kernel Tuning Toolkit)[1] - OpenCL and CUDA. Because both APIs provide relatively similar functionality, only OpenCL is described here in greater detail. Section about CUDA is mostly focused on explaining features which differ from OpenCL. It is worth mentioning that CUDA actually consists of two different APIs - high-level runtime API and low-level driver API. For the purpose of this thesis, only CUDA driver API will be further described, because the runtime API lacks features which are necessary to implement autotuning in CUDA.

2.1 OpenCL

OpenCL is an API for developing primarily parallel applications which can be run on a range of different devices such as CPUs, GPUs and FPGAs (field-programmable gate arrays). An OpenCL application consists of two main parts. First part is a host program, which is typically executed on a CPU and is responsible for OpenCL device configuration, memory management and kernel execution. Second part is a kernel, which is a function executed on an OpenCL device and usually contains major part of a computation. Kernels are written in OpenCL C which is a language based on C programming language.

2.1.1 Host program in OpenCL

Host program is written in regular programming language, typically in C or C++. Its main objective is to successfully launch a kernel function. OpenCL API defines several important structures which are referenced from host program:

- cl_context - serves as a holder of resources, similar to OS process, majority of other OpenCL structures have to be initialized inside a specific context

1. Name of the autotuning framework developed as a part of the thesis.

- cl_command_queue - all commands which are executed directly on OpenCL device have to be submitted inside a command queue, it is possible to initialize multiple command queues within a single context in order to overlap independent asynchronous operations
- cl_buffer - todo...
- cl_kernel - todo...
- cl_program - todo...
- cl_event - todo...

Execution of an entire OpenCL application then typically consists of the following steps:

- selection of target platform (eg. AMD, Intel, Nvidia) and device (eg. GeForce GTX 970)
- initialization of OpenCL context and one or more command queues
- initialization of OpenCL buffers (either in host or dedicated device memory)
- compilation and execution of kernel function
- transfer of data produced by kernel from OpenCL buffers into host memory (if data is located in dedicated device memory)

2.1.2 Kernel in OpenCL

Todo...

2.2 CUDA, comparison with OpenCL

Todo...

2.3 Autotuning in compute APIs

Todo...

3 Conclusion

Todo...

Bibliography

[1] Cedric Nugteren and Valeriu Codreanu. "CLTune: A Generic Auto-Tuner for OpenCL Kernels". In: *MCSoC: 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. 2015.

[2] Khronos OpenCL Working Group. *The OpenCL 1.2 Specification*. 2012. URL: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf (visited on 02/25/2018).

A Appendix

Todo...