Documentation for the JuliaGPU stack

Introduction

With the end of Moore’s law and Dennard scaling our quest for performance has led us to invest in various technologies like multicore CPUs, SIMD units, graphical processing units, field programmable gate arrays, application specific integrated circuit,. etc in the past few decades. These specialized technologies are designed with specific workloads in mind presenting their own set of challenges, advantages and disadvantages. One such challenge is the development of new infrastructure such as compilers, debuggers,.etc which are compatible with these new technologies. Another challenge is to educate programmers on how to effectively use these new technologies to save time, capital and energy.

A GPGPU (general purpose graphical processing unit, henceforth called GPU) is a hardware accelerator which has been extremely popular for the past decade. It is used in various domains like graphics, data science and machine learning. The main theme behind a GPU is to do parallel processing by distributing the work among a large number of parallel threads executing on a large number of primitive cores. In contrast a modern CPU has a small number of very powerful cores. The GPU programming is based on the SIMT (simgle instruction multiple thread) model whereas the traditional CPU model is called SISD (single instruction single data).

An issue with the GPU is that regular CPU code cannot be executed on it. A GPU has a different instruction set hence requires some modification to source code, a different compiler, assembler,.etc. Since the main aim of using a GPU is to achieve higher performance it is important for developers to understand how a GPU works. GPU code requires different algorithms and paradigms. To a beginner, who presumably is only used to regular programming this task can be really difficult.

Julia is a high-level programming language which aims to solve the two language problem. The two language problem refers to the prototyping and deployment of software in different languages. The reason for this is the commonly “understood” phenomenon of developer productivity to execution speed tradeoff. Prototyping is often done in a higher level language such as Python for productivity and production code is rewritten in a low level language such as C++ for performance. Julia aims to solve this problem with the help of a clever type system, a robust compilation strategy and many more features.

For the past couple of years a GPU stack for Julia has been under heavy development. It allows developers to develop in native Julia code in contrast to CUDA C or OpenCl which is very low-level and almost C/C++ code. With the CUDA stack (CUDA.jl) being mature at this point it is a great time to start developing GPU software using Julia.

Aim

The primary purpose of this project is to develop documentation rich with tutorials and examples to help developers get started with GPU programming in Julia. Unfortunately, there aren't many resources for GPU programming, and being a relatively complex topic it gets difficult for the learner. To use a GPU effectively not only must a user know how to use it but also understand how it works

This project can be split into three parts (not proportional to time)

- Tutorials: Detailed guides for the beginner to help them get started, profile and debug their code.
- Examples: Write simple extensions to various Julia packages such as Images.jl, SciML.jl,etc.
- API Documentation: Restructure and write documentation to make it intuitive for the user to browse through the documentation.

While this project focuses on CUDA.jl it shouldn't matter since to a large extent Julia's GPU stack is going to be platform agnostic so in the future when the AMD's ROCm stack is completed and the Intel GPUs arive there won't be much effort required to port the Tutorials and Examples of this project.

The tutorials and examples are proposed to go in a dedicated section at the juliagpu.org website. Some topics which will be covered in the tutorial section are:

- Introduction (small rework)
- Mandelbrot : A program to generate an image of the mandelbrot set
- Prefix Scan: Computing the parallel prefix scan on the GPU
- How does a GPU work ? (language agnostic, discusses the architecture)
- Array Programming: Using high level array programming abstractions for GPU programming (Broadcast abstractions, custom array types,.etc)
- Profiling GPU applications (using Nsight and other tools)

Tools

The tutorials and examples will be written as Jupyter notebooks and converted to HTML using `Weave.jl` in the same way as `DiffEqTutorials.jl`.

The API Documentation will be done in the same way as now which uses `Documenter.jl`.

Application Details

Project Title: Documentation for the JuliaGPU Stack

Prospective Mentors: Tim Besard and Valentin Churavy

Github handle: Ellipse0934