NAME

Contributing to Futhark for your bachelor's project.

SYNOPSIS

Applied and somewhat challenging projects on parallel programming and compilers, where you will build something that might actually be used.

DESCRIPTION

Futhark is a parallel programming language developed by a research group in the PLTC section. It is used conduct research into compiler construction, programming language design, and parallel programming. Futhark is developed as an open source project <https://github.com/diku-dk/futhark>. and sees use both inside and outside DIKU for practical programming, as it allows programmers to easily write high level programs that run very fast on GPUs. Throughout its development, student projects at all levels have contributed many improvements. <https://futhark-lang.org/publications.html#selected-student-projects>. Futhark itself looks like a simplified combination of Standard ML, Haskell, and F#:

```
def vector_add [n] (a: [n]i32) (b: [n]b): [n]i32 =
  map2 (+) a b
def product [n] (a: [n]i32): i32 =
  reduce (*) 1 a
def dot_product [n] (a: [n]i32) (b: [n]i32): i32
  product (vector_add a b)
```

You are not expected to know Futhark in advance to do any of our projects, but it is in any case not a big language.

The projects listed below all involve working on or in close proximity to the Futhark compiler itself. That is, they are *applied* projects. Not all of them involve direct compiler work - some are about working on related tooling - but they typically involve the kind of programming that you were exposed to in your course on implementation of programming languages. The Futhark compiler is written in Haskell, which is in many ways similar to F#. If you don't already know Haskell, that is not necessarily a problem - previous students have learned it along the way and conducted successful projects. Some projects may also involve working on run-time systems, which are typically written in C. The projects listed here are *somewhat difficult* because they involve building something that has to *actually work* in a real environment. This means they are not suitable for weak programmers. On the upside, there is a good chance that the work you do will actually be used in practice, by various people around the world. Also, since all of these projects are about work that we are interested in having available for our own use, you will receive as much supervision as necessary. Finally, they provide a good foundation for doing more involved compiler hacking projects later on.

PROJECT: Automatic Differentiation in the Futhark interpreter

Modern machine learning relies on Automatic Differentiation (AD); a technique for computing derivatives of arbitrary programs. We are implementing an advanced highly optimised AD transformation in the Futhark *compiler*, but we would also like to support AD in the Futhark *interpreter*. This is partly because the interpreter is much more useful for interactively debugging Futhark programs, and partly because a "known correct" implementation would be useful for testing the AD implementation in the compiler.

More Information

https://github.com/diku-dk/futhark/issues/1556

PROJECT: Automatic Code Formatting

Tools such as "go fmt", "rustfmt", and "black" have become popular for automatically (re)formatting source code. We would like to also have such a formatter for Futhark. Probably this will involve modifying the existing lexer and parser to provide the necessary information, and studying formatters for other languages to understand how they work. This is a project with potentially high impact, since most Futhark programmers would be using this tool very frequently.

PROJECT: Porting benchmarks from PBBS

To evaluate Futhark's performance we *port* programs that are already known to be efficiently written in some other language and see how fast they run in Futhark. We have a growing collection of benchmark programs written in Futhark. <https://github.com/diku-dk/futhark-benchmarks> To find more programs worth porting, we look at existing *benchmark suites* written by others. One such suite is the Problem Based Benchmark Suite <https://cmuparlay.github.io/pbbsbench/> developed at CMU, from which we currently have a number of ports <https://github.com/diku-dk/futhark-benchmarks/tree/master/pbbs>.

This project is about picking one or more of the problems in PBBS that we currently miss, porting them to Futhark, and measuring their performance compared to the original implementation. The project also involves qualitatively evaluating Futhark's suitability for the problem in question.

Some of the particularly interesting problems are the ones related to graph algorithms or computational geometry. This project is particularly suited for students who are not interested in compiler construction, but very interested in parallel programming.

PROJECT: WebGPU Backend

The Futhark compiler currently has two GPU backends, targeting the CUDA and OpenCL APIs. This

project is about adding a backend for WebGPU, which would allow Futhark programs to run in a browser. The basic programming model for all of these APIs is the same, where a CPU-side program sends code and data to the GPU, but the details differ. The project involves two somewhat independent parts:

- Generation of *WebGPU Shading Language* (WGSL) compute shader code from the Futhark compiler's current kernel representation. This is hopefully a straightforward adaptation of the existing OpenCL and CUDA kernel generators, but it is likely that WGSL has some constraints or limitations that will require some creativity.

- Generation of the CPU code that makes use of the WebGPU API. Futhark has a WebAssembly backend (thanks to a prior student project), and our hope is that this can be used as a foundation through appropriate libraries this <https://github.com/juj/wasm_webgpu>.

Beware

This is a very challenging project that will guarantee you a seat in Valhalla. Equally challenging bachelor projects have succeeded before, but in order to complete this project, you *must* be a strong programmer (and ideally form a group with other strong programmers) and be able to quickly understand and resolve complicated technical challenges. However, you do not need to know anything about GPU programming in advance.

PROJECT: Static Analysis of Irregular Parallelism

The Futhark compiler currently supports *regular* nested data parallelism, which is a form of parallelism where the size of inner parallel dimensions are invariant to the outer dimensions. Even though we are working on supporting arbitrary irregular nested data parallelism, this will never be as efficient as regular nesting. Unfortunately, whether application parallelism is regular or irregular is an implicit property, and it can be hard for programmers to know when it occurs.

This project is about writing a static analysis tool that takes as input a Futhark program and analyses it for potential instances of irregular nested data parallelism. This is a fun project if you enjoy working with ASTs and devising algorithms.

PROJECT: Implementing Iverson's Derivative Operator

Turing Award winner Ken Iverson is famous for his contributions to computer science, in particular through creation of the APL programming language. Although APL uses an infamously cryptic notation by modern standards, its semantics for bulk operations of arrays are largely the core of modern libraries such as NumPy and its imitators.

In 1979, Ken Iverson defined the *derivative operator* <https://dl.acm.org/doi/pdf/10.1145/800136.804486> for taking the mathematical derivatives of APL functions. Similar operators have become increasingly important, particularly for machine learning, where the automatic generation of derivatives is a ubiquitous tool. It would be interesting to investigate the suitability Iverson's original design compared to modern automatic differentiation (AD) systems.

This project is about extending the DIKU-developed APL-to-Futhark compiler tail2futhark <https://github.com/henrikurms/tail2futhark> with support for Iverson's derivative operator, on top of Futhark's existing support for AD. One important initial step is to update tail2futhark to support the current Futhark language.

Note

Martin Elsman <mael@di.ku.dk> will be the main supervisor for this project.

CONTACT

Troels Henriksen <athas@sigkill.dk> if you have questions, or alternatively Cosmin Oancea or Martin Elsman.

BUGS

Troels strongly prefers supervising groups, as the failure rate for single students is too high. Exceptions can be made for particularly motivated students. Note that *desperation* is not equivalent to motivation.