Contributing to Futhark for your Bachelors Project

Troels Henriksen (athas@sigkill.dk)

Computer Science
University of Copenhagen

September 2019

Contribute to the best data-parallel GPU-targeting ML-like functional language developed at DIKU!

Looks a bit like a simplified combination of SML and Haskell.

```
let vector_add [n] (a: [n]i32) (b: [n]b): [n]i32 =
  map2 (+) a b

let product [n] (a: [n]i32): i32 =
  reduce (*) 1 a

let dot_product [n] (a: [n]i32) (b: [n]i32): i32
  product (vector_add a b)
```

It runs *very fast* on parallel hardware. Significantly faster than most hand-written C or similar.

Roughly Three Kinds of Projects

(Light) Compiler Hacking:

The Futhark compiler is written in Haskell. It is well written, but not small (55k SLOC). Only fairly limited work possible for a bachelors project.

Parallel Programming in Futhark:

We port benchmarks, libraries, and example applications to determine areas in which Futhark should be improved.

Work on Infrastructure and Tooling:

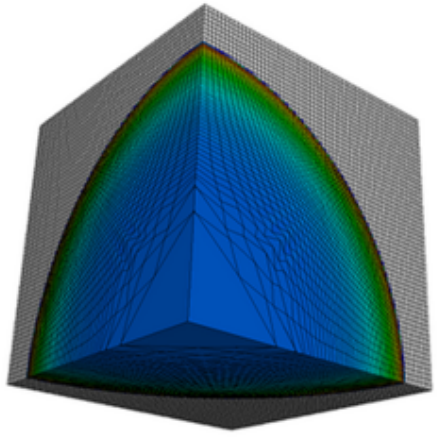
Futhark is run as a sound software engineering project, which means tooling to run tests, analyse benchmark results, etc. These are very "open" projects, in that you are free to choose language and methodology yourself.

The Unit of Difficulty

- We rate the difficulty of project proposals from one to five broken keyboards.
- Not simply a measure of workload, but based on whether we believe we understand every issue related to implementation of the project.
- Higher difficulty implies more "unexpected" problems that we cannot immediately give you an answer to.

Project: Porting LULESH to Futhark

Idea LULESH a small fragment of a program for simulating shockwave propagation, commonly used for showing off new languages and tools.



Challenges Nontrivial size (the Accelerate port is over a thousand lines).

Difficulty 

While I don't understand the physics myself, there are dozens of existing implementations.

Project: Porting LULESH in Futhark (details)

To quote LLNL:

> *"The Shock Hydrodynamics Challenge Problem was originally defined and implemented by LLNL as one of five challenge problems in the DARPA UHPC program and has since become a widely studied proxy application in DOE co-design efforts for exascale."*

In the HPC community, porting or running LULESH is a fairly common way of showing off new languages or machines. While Futhark is not really targeted at exascale computers, we do have a strategy of porting common benchmarks to show off what the language can do. LULESH already has many implementations in different languages, both high- and low-level, so you don't need to understand the physics to port it. It is generally the case when porting benchmarks that we don't actually know what the program *does*, we just want to make sure our version gives the same result.
This is a hacking-heavy project, and fun if you like making things go fast, or just want to try out real parallel programming on a nontrivial problem.

Project: Investigate and Improve C# Backend

Idea We recently finished a succesful master's thesis project that implemented a C# backend for Futhark. This project also includes a compiler from a small subset of F# to Futhark. We think it works well. We might be wrong. We'd like someone to investigate more closely.

Challenges You'll be working with fairly new technology that is pretty untested, even by the standards of research language.

Difficulty

It is very easy to scale this project based on your ambitions, skills, and interests.

Project: Investigate and Improve C# Backend

The C# backend can produces standalone executables, just like the other backends. This has been tested quite well, but is useful only for benchmarking. The ambition of the C# backend is that the generated code can be accessed by otherwise ordinary C# applications, but this has not been evaluated much. This project could investigate how easy it is to take existing C# programs with some computational component, and rewrite that component in Futhark. Alternatively, it could take one of the existing Futhark demos[1][2][3], which are written in Python and Go, and rewrite them in C#.
We also have a compiler, FShark, from a subset of F# to Futhark. The point here is that an F# programmer can develop and debug F# code as they are used to and with conventional tools (subject to some extra rules), and then later compile it to Futhark, and from there to efficient GPU code. We have no idea how practical this workflow is. Maybe you can find out for us?

The overarching purpose of this project is to produce feedback for improving these two bits of technology. If you

are up to it, the project can also involve actually *performing* said improvement, but it is not a requirement.

---

[1] `https://github.com/athas/diving-beet`
[2] `https://github.com/athas/futball`
[3] `https://github.com/nqpz/futcam`

Project: Javascript Backend

Idea Javascript does not have any OpenCL binding (WebCL is dead). Instead, we could use *compute shaders* from the OpenGL binding - WebGL.

Challenges I have no WebGL experience; our kernel code generator only generates OpenCL kernels (although GLSL shaders look fairly similar); requires *significant* amounts of research, design and implementation work. This is definitely a tough one.

Difficulty

Do not assume that you can finish this completely in one project - think proof of concept. I believe this project constitutes novel research.

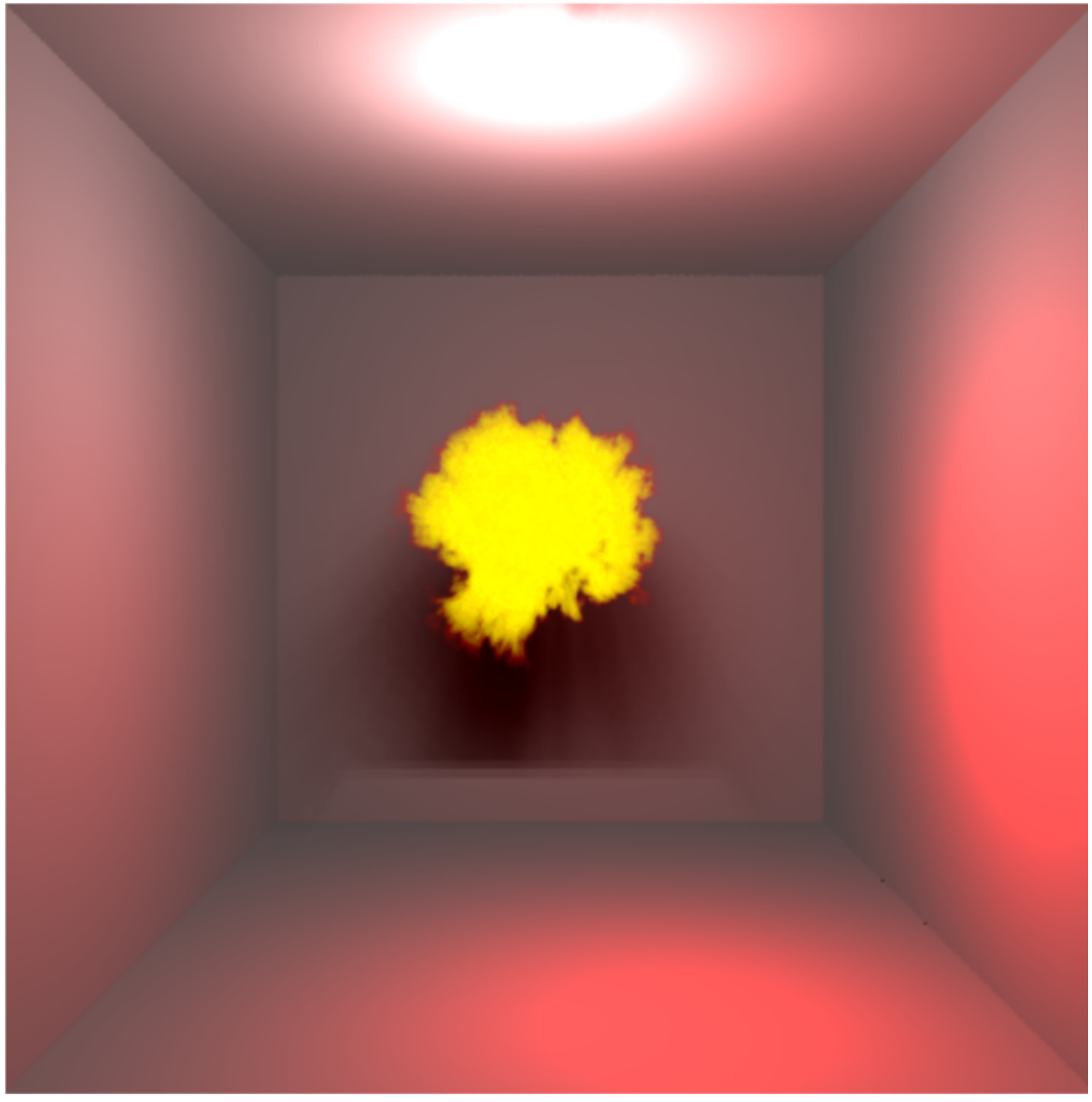
Project: Implement Hypertexture Rendering

Idea Hypertexture is a method for representing 3D models with complex boundaries, like fur or fire. Rendering hypertexture scenes is very time-consuming, but very parallel. A combinator-based library for expressing hypertexture scenes in Futhark would be fun.

Challenges You will need to understand the underlying mathematics to at least some extent. Designing a library will require thinking hard about API design (especially given Futhark's constraints).

Difficulty

Project: Implement Hypertexture Rendering (image)



Project: Implement Hypertexture Rendering (details)

Samantha Frolich from the University of Bristol has implemented hypertexture as a Haskell library, which then generates a C++ program to perform the rendering. She won the ICFP undergraduate research competition for her work, which is now publicly available:
`https://github.com/sf16540/Hypertexture`
The scheme of generating C++ is necessary because Haskell is nowhere fast enough for the actual rendering (based on raymarching) to run in reasonable time. Hopefully, Futhark is!
Samantha's implementation is rather complex and heavily based on type classes, allowing her implementation to be instantiated with different algebras. I don't think this is crucial to the idea, and in Futhark it is probably better to just design a library that ultimately represents a *fuzzy set* (so, a probabilistic point distribution). This is not too dissimilar to when we ported Conal Elliott's *functional images* to Futhark with very good results:
`https://github.com/diku-dk/futhark-benchmarks/tree/master/misc/functional-images`

Project: Bounds Checking on GPU

Idea The Futhark compiler makes an effort to remove array bounds checks, but it is not always succesful. Currently, the GPU code generator cannot deal with bounds checks, so if any are left, the compiler will fail. The workaround is to disable bounds checking for some programs, but this is not very satisfactory. This project is about finding a way to efficiently do bounds checking on the GPU.

Challenges Need to find a way to abort a GPU thread without deadlocking the entire kernel (or catch the deadlock, or kill the entire kernel somehow). The failure case need not be fast, but passing checks should be efficient.

Difficulty

Other Projects

If you have an idea of your own, come by and we can have a chat about it. Inspiration:

- A "Try Futhark" web application like `https://tryhaskell.org/`.
- Port a benchmark from a suite like Rodinia or Parboil to Futhark.
- Writing a Futhark package or application that does something cool: classification, machine learning, image analysis, etc.
- An automatic code formatter like `gofmt`.
- Fixing/improving/rewriting the automatic indentation of the `futhark-mode` for Emacs.

If you have an ≤ 15 ECTS idea for something that would contribute to the project, we can probably supervise it.

Compiler hacking is hard work, but rewarding

"Futhark: the harder the battle the sweeter the victory!"
-Rasmus Wriedt Larsen (bachelor and masters thesis)

"Hvis du laver et Futhark-projekt, kommer du aldrig til at sove."
-Niels G. W. Serup (bachelor and masters thesis)

"Et ambitiøst projekt med en hårdt arbejdende kerne af folk! Hvad mere kan man ønske sig?" *-Hjalte Abelskov (bachelor thesis)*

"Efter at havde arbejdet med Futhark føles alt andet i mit liv meningsløst." *-Daniel Gavin (bachelor thesis)*

"Futhark: Så lille et sprog, der kan så meget for så mange!"
-Kasper Abildtrup Hansen (bachelor thesis)

Contact

If you think any of this is interesting, come talk to Troels...

- ...in my office: 01-0-017 at HCØ.
- ...via email: `athas@sigkill.dk`
- ...or on IRC: #diku on Freenode (`http://ucph.dk/#tabs1-chat`)

Or talk to Martin Elsman or Cosmin Oancea.

Also check out the website at `https://futhark-lang.org`