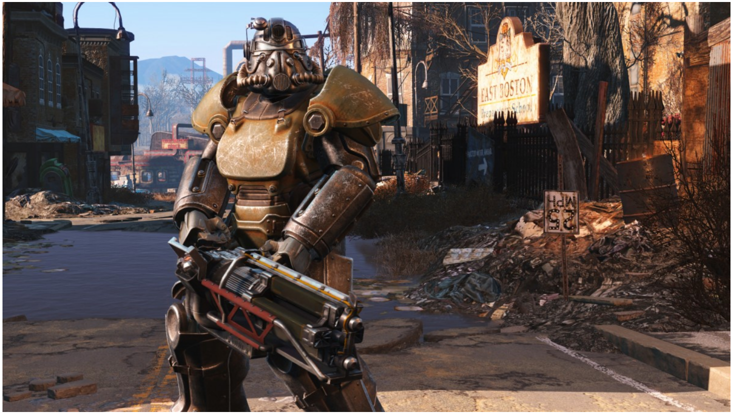
Hacking on the Futhark Compiler for your Bachelors Project

Troels Henriksen (athas@sigkill.dk)

Computer Science
University of Copenhagen

8. February 2016

A Computer Scientist is a person who gets his hands on a nice new graphics card, not to look at this:



But at this:

```
clGetPlatformIDs(1, &platform, &platforms);
clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1,
               &device, &devices);
cl_context_properties properties[]= {0};
cl_context context =
  clCreateContext(properties, 1, &device,
                  NULL, NULL, &error);
cl_command_queue cq =
  clCreateCommandQueue(context, device,
                       0, &error);

cl_program prog =
  clCreateProgramWithSource(context, 1,
                            &transpose_cl,
                            NULL, &error);
clBuildProgram(prog, 0, NULL, "", NULL, NULL);
```

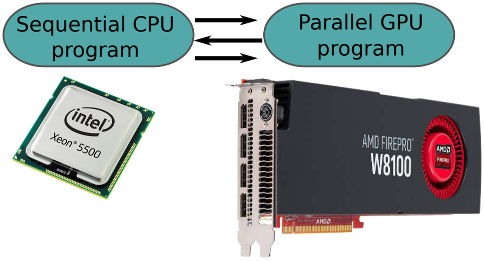
Context

- Graphics cards (*GPUs*) provide massive amounts of computational power, but they are hard to program. It gets even harder (impossible) if you want both reusable and fast code.
- Still, GPGPU (using GPUs for non-graphics workloads) has been increasing in popularity for the past ten years, as the potential performance improvements are significant.
- GPUs are extremely parallel processors capable of keeping tens of thousands of threads in flight. But these threads have to access memory in certain patterns, and do roughly the same, or they will run very slowly.
- This calls for a language that makes it easier to deal with.

GPU programming

GPUs function as extremely fast data-parallel *co-processors*. They are not independent, but are controlled by an ordinary CPU process that sends code and data.

- For strange historical reasons, GPU programs are often called *kernels*. The CPU code is called *host code*.
- At runtime, a CPU program will send a GPU program (often specified in a dialect of C) to the device driver for the GPU, and get back some GPU-specific machine code.
- This GPU-code is then sent to the GPU along with data in order to perform computation.

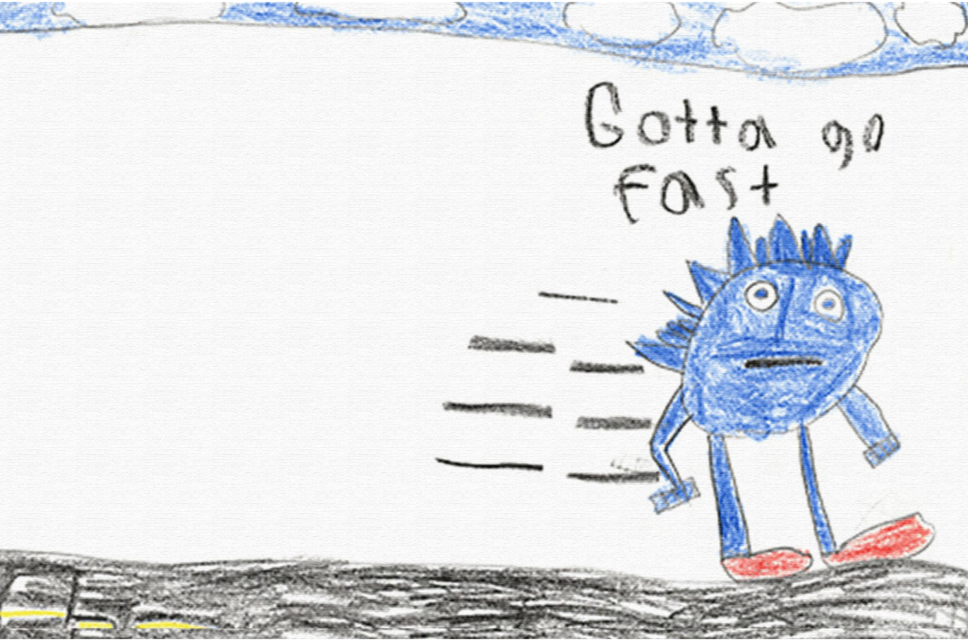


Futhark overview

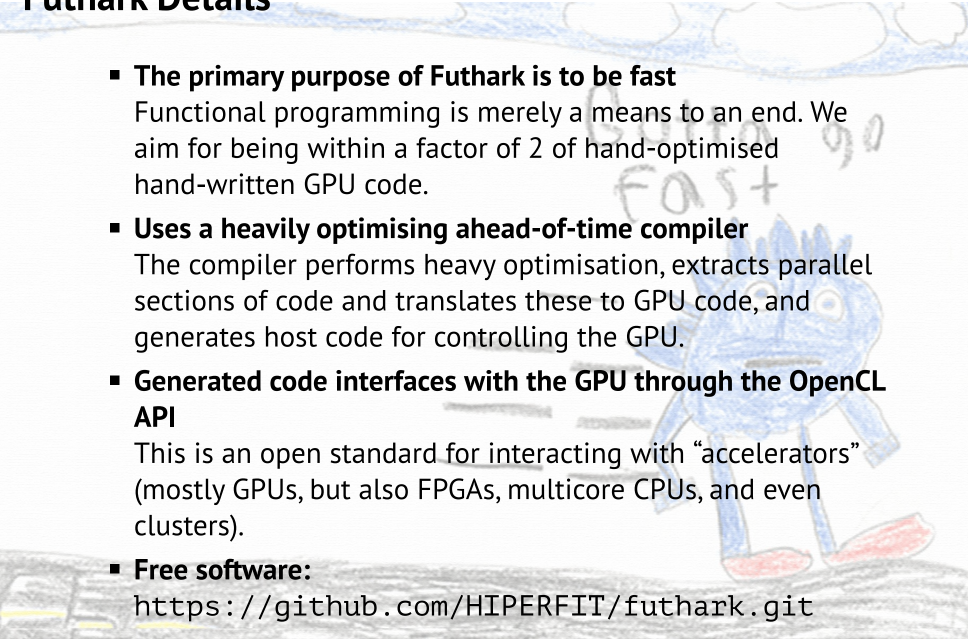
We have created a programming language, *Futhark*, intended for writing parallel programs that can be compiled to run on GPUs. Futhark is a data-parallel purely functional language. Looks a bit like a simplified combination of SML and Haskell.

```
fun [int,n] sumPairs([int,n] a, [int,n] b) =
  zipWith(+, a, b)

fun int product([int] a) =
  reduce(*, 0, a)

fun int dotProd([int,n] a, [int,n] b) =
  product(sumPairs(a, b))
```

So what do we want to do with this language?


Gotta go
Fast

Futhark Details



- **The primary purpose of Futhark is to be fast**
  Functional programming is merely a means to an end. We aim for being within a factor of 2 of hand-optimised hand-written GPU code.
- **Uses a heavily optimising ahead-of-time compiler**
  The compiler performs heavy optimisation, extracts parallel sections of code and translates these to GPU code, and generates host code for controlling the GPU.
- **Generated code interfaces with the GPU through the OpenCL API**
  This is an open standard for interacting with “accelerators” (mostly GPUs, but also FPGAs, multicore CPUs, and even clusters).
- **Free software:**
  `https://github.com/HIPERFIT/futhark.git`

Projects Overview

- Most of the projects involve modifying parts of the Futhark compiler, which is written in simple Haskell. If you know SML, you can quickly pick up enough Haskell to be dangerous.
- Some projects also require low-level programming effort in C or C++.
- *All* projects require significant practical implementation work.
- *No* projects are a waste of time - they are all worthwhile things we want or need for research.
- I strongly suggest you work in groups - compiler work is more time-consuming than it looks.

Some projects have more details at

`http://hiperfit.dk/studentprojects.html`

The Unit of Difficulty

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

Project: New Host Language Backends

Idea The host code could be generated in any language with OpenCL bindings. Since most of the runtime is ideally spent in the GPU code, performance of the host language should not matter. We already have code generators for Python and C - one could improve these, or add Ruby, Java, C#, F#, C++, Haskell, or whatever happens to be your favorite language!

Challenges Mapping Futhark IL constructs and types to the target language; generating a nice API.

Difficulty

We have basically done this before - but it’s still fun and rewarding work.

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: Benchmark Infrastructure

Idea Create an automatic system that can run our benchmark suite, collect performance statistics, store them in a database (or just data files), and produce online graphs of changes as we work on the compiler. Should run automatically every now and then.

Challenges Involves a bit of independent problem analysis to figure out what we need, as well as some UI design and visualisation work. You can probably look at established compiler projects for inspiration.

Difficulty ⌨ ⌨

Feasible even for a single student - and you can pick whichever language, infrastructure and tools you want.

Project: CUDA Backend

Idea	Currently, the Futhark compiler generates code using the OpenCL library, but some interesting platforms support only NVIDIAs CUDA. It would be useful to add a CUDA backend to Futhark.
Challenges	There are three different CUDA layers that could be targed by the code generator: language, runtime, and driver level. I have no idea which one is best. Will possible some minor modification of kernel code generator (the CUDA and OpenCL kernel languages are similar but not identical).
Difficulty	⌨ ⌨ ⌨

Project: Compile-Time Memory Allocation

Idea	Late in compilation, the Futhark compiler assigns each array to a memory block. Currently, each array gets a unique memory block, but several arrays could use the same block, as long as the arrays are not *live* at the same time. This project is about implementing liveness analysis and an algorithm similar to *register allocation* to do this in the Futhark compiler.
Challenges	Need to grok the Futhark memory representation. The immediate goal (lowered memory usage) probably easy to reach; but reducing amount of copies may be harder.
Difficulty	⌨ ⌨

Project: Loop-Scan Loop Fusion

Idea	Efficient parallel execution typically involves every thread computing a chunk sequentially, followed by the threads combining their results. Good performance depends on taking advantage of this sequential stage, for exampl by fusing `map` and `scan` operations (which is not valid in the parallel stage). We already do this for `map` and `reduce`, where it works well.
Challenges	You have to modify the loop fusion pass in the Futhark compiler, which is a bit gnarly. This is still a gentle introduction to high-level loop optimisations.
Difficulty	

Project: Function-Level Shape Invariants

Idea This project is about allowing the user to annotate, at function level, algebraic invariants between integer parameters, which for example, may allow the compiler to make optimisation decisions and bounds check removal. For example:

```
fun int myFun(int N, [[real, k4],M] arr)
          where ( k1 < k2,
                  k3 in k2*k1 + N ... M,
                  k4 < M + N,
                  M >= max(2*N,1),
                  arr in N ... M-1 ) =
   body of function
```

Challenges Involves language design; not clear which invariants are the most useful.

Difficulty

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 globally 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). Bounds checking errors need not be fast, but passing checks should be efficient.
Difficulty	⌨ ⌨

Project: Multicore Backend (OpenMP)

Idea	Futhark development has mostly focused on GPUs, but modern multicore CPUs are also a good fit for data-parallel code. We already have a good sequential code generator, and it should be doable to use something like OpenMP to generate parallel CPU code.
Challenges	Some parts of the compilation pipeline make performance assumptions that are only valid on GPUs. This project will require small changes in all steps of the compiler, which means you need to understand a largish code base.
Difficulty	⌨ ⌨ ⌨ ⌨

Other Projects

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

- **Futhark Interpreter** with a focus on debugging Futhark programs (breakpoints, single-stepping, variable inspection, etc).
- **Translating benchmark programs to Futhark**, with a focus on looking for new language features that would be useful when writing high-performance code.
- **Syntax improvements**, such as permitting expressions like `map(map(f), a)` – currently the Futhark parser is quite crude, and does not permit currying of control flow structures.
- **Compiling some other language to Futhark**, for example some domain-specific language. We already had one (very nice) bachelor's project about compiling APL to Futhark.

Compiler hacking is hard work, but rewarding

"At arbejde med Futhark var både spændende og lærerigt."
-Maya Saietz

"Hvis du laver et Futhark-projekt, kommer du aldrig til at sove"
-Niels G. W. Serup

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

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

Contact

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

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