Final project ideas

CS448h

Oct. 22, 2015

Tools for building DSLs

Investigate ways of **debugging DSLs**. How can domain-specific knowledge let you make a richer debugger?

Build a **high-level profiler** for Terra (sampling-based or performance counters)

See Lua debug module for design ideas.

Build some **general optimization modules** for building DSLs. For example, an algebraic simplifier, a rewrite system, ...

Tools for building DSLs

Implement an **OMeta**-inspired DSL in Lua/Terra to **make implementing DSLs easier**

(maybe show that their design can successfully be used to simplify the implementation of some existing DSLs) (Is it better to just use vanilla Lua?)

Build a **Terra-like** system for **Python** (or Javascript)

Investigate different ways of doing **syntactical macros**

Build a new DSL

Database-inspired DSL
Vector or array processing DSL
Linear algebra DSL
Tensor-algebra DSL

Compile graphical models, or another class of machine learning problem

Bonus: fast inference on a GPU.

Build a new DSL

Implement an in-memory “database" that other DSLs can use for their implementation.

i.e., the challenge is that something like Halide, Liszt, Opt, … could all just use this one abstraction, possibly to interoperate?

revisit “Programmable Rendering of Line Drawing from 3D Scenes” as a kind of NPR DSL.

Given better DSL technology, can it be used for real-time rendering? Can the interface/language be improved?

Extend an existing DSL

DSL composition: choose two interestingly different DSLs and make them interoperate/compose.

Ebb + Halide could be an obvious choice.

Explore the struct-of-arrays vs. array-of-structs tradeoff in an existing language

more generally, experiment with methods for laying out memory based on the code

Build domain-specific IDE features for a DSL

Extend an existing DSL: Halide

JIT a library: implement underneath the OpenCV abstraction, and generate better code dynamically for pipelines/compositions of operations

Automatically infer good schedules

Extend an existing DSL: Lizt/Ebb

Render directly from simulations.

Specialized shading language?

Liszt-Ebb interface for writing linear operators.

Use this to define a standardized interface to linear solver libraries.

Automatic differentiation for Liszt-Ebb as a way to specify/construct linear operators at a given state.

Explore possible Kernel fusion-fission rewrites.

Extend an existing DSL: Lizt/Ebb

Data abstraction. Create a higher-level relational model that compiles down to the Ebb model by choosing primary indices, etc.

Maybe this could also handle memory-efficient layouts of symmetric/anti-symmetric stiffness matrices, etc.