Ikra: Leveraging Object-oriented Abstractions in a Ruby-to-CUDA JIT Translator

Matthias Springer, Hidehiko Masuhara
(Tokyo Institute of Technology)

Overview
- Acceleration of Ruby programs with GPUs (CUDA)
- High-level Goal: GPGPU for Ruby programmers
- Source code analysis and type inference at runtime

Restrictions in parallel sections:
- No restrictions
- No meta programming/reflection outside of dynamic scope
- No dynamic typing should be avoided
- Parallel sections

- High-level Goal: GPGPU for Ruby programmers
- Acceleration of Ruby programs with GPUs (CUDA)

Object Support
- Allocating threads: only objects of the same class per warp
- Improve code quality using class-based programming

TrafficSimulator
- Actor
  - @max_velocity
  - @progress
  - @street
- TrafficSimulator
  - Car
  - Pedestrian
- TrafficSimulator
  - move

Heap Object Tracer
- 1. Type inference
- 2. Find objects and calculate column offsets
- 3. Write object columns

Kernel Fusion
- Job reordering
- 1. Merge cascaded commands (kernel fusion)
- 2. Infer types and whether instance variables are read or written
- 3. Generate CUDA code
- 4. Compile shared library (nvcc)
- 5. Reorder jobs (avoiding warp divergence)
- 6. Trace reachable objects, allocate and transfer objects (Ruby FFI)
- 7. Invoke kernel
- 8. Write back written columns

Columnar Object Layout
- Idea: Represent all objects of a class as fields of arrays (columnar layout)
- Benefit: Chance for coalescing when accessing the same column in parallel
- Implementation: Heap Object Tracer converts object graph to columnar layout