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

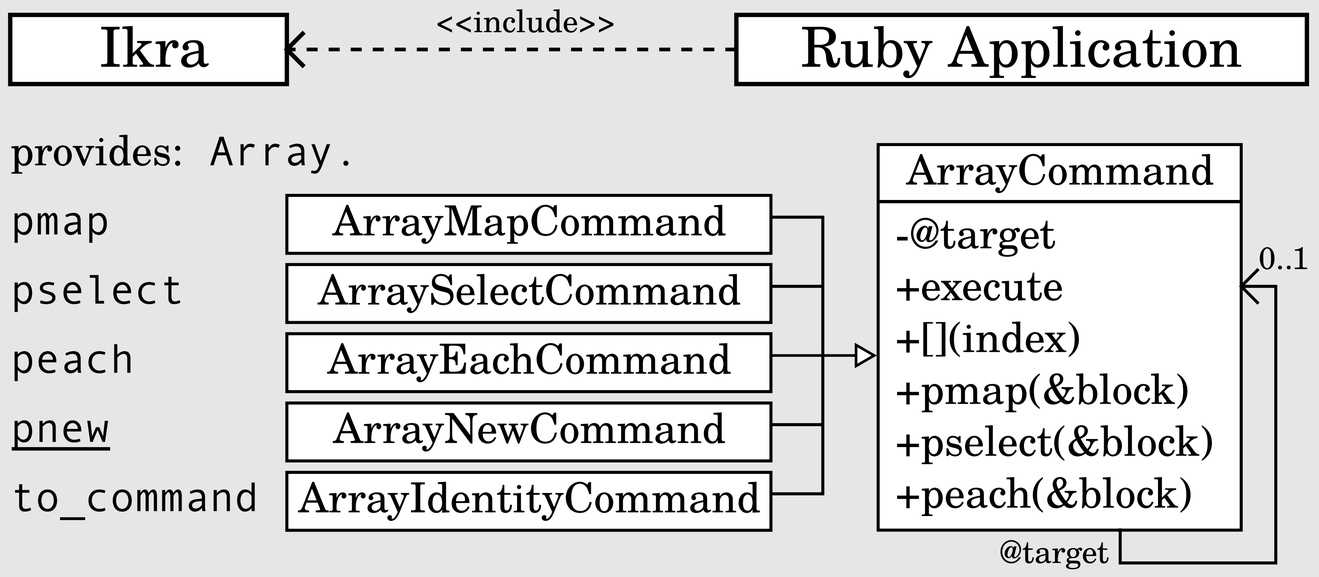
Restrictions in parallel sections:
- No meta programming/reflection
- Dynamic typing should be avoided

No restrictions outside of parallel sections

Related Work:

- Job Reordering
  Zhang et al. On-the-fly elimination of dynamic irregularities for GPU computing. ASPLOS XVI.
- Kernel Fusion
  Wahib et al. Scalable kernel fusion for memory-bound GPU applications. SC '14.
- Columnar Object Layout
  Mattis et al. Columnar objects: improving the performance of analytical applications. ONWARD! 2015

Architecture

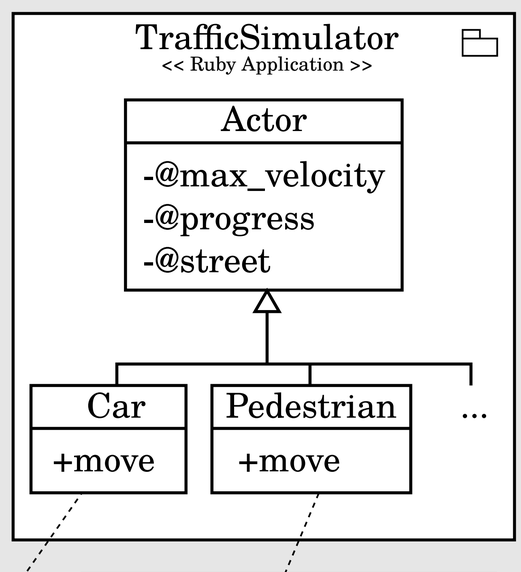


Overview:
1. Merge cascaded commands **(kernel fusion)**
2. Infer types and whether instance variables are read or written
3. Generate CUDA code
4. Compile shared library (`nvcc`)
5. Reorder jobs (avoiding **warp divergence**)
6. *Trace* reachable objects, allocate and transfer objects (Ruby FFI)
7.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

Object Support

```
actor_size = 20
actor_type = [CAR, CAR, ...]
actor_max_velocity = [...]
# ...

def run_simulation
   weather = UI.selected_weather

   (1..actor_size).peach do |i|
      if actor_type[i] == CAR
         move_car(i, weather)
      elsif actor_type[i] == PED
         move_ped(i, weather)
      end
   end
end
```

warp divergence

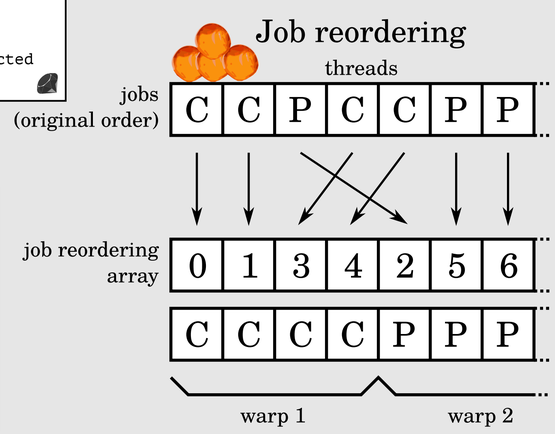



```
@progress += Random.rand(@max_velocity)

if @progress >= @street.length
   @street = @street.random_connected
end
```

with objects

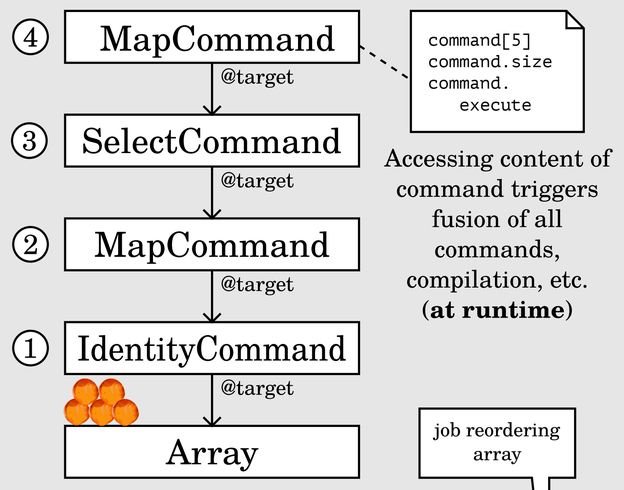
```
if weather == BAD
   @progress += 0.5 * @max_velocity
else
   @progress += 0.95 * @max_velocity
end

if @progress >= @street.length
   @street = @street.random_connected
end
```

```
actors = [...]

def run_simulation
   weather = UI.selected_weather

   actors.peach do |actor|
      actor.move(weather)
   end
end
```



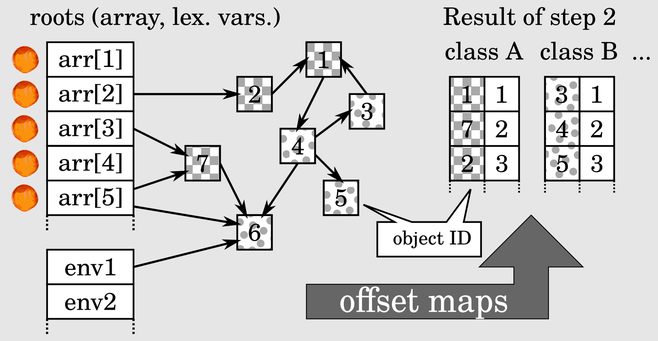
- Allocate threads: only objects of the same class per warp
- Improve code quality using class-based programming

Kernel Fusion




```
__device__ int block_1(int el) { return el; }
__device__ int block_2(int el) { return 2 * el; }
__device__ bool block_3(int el) { return el > 10; }
__device__ int block_4(int el) { return el + 1; }

__global__ void kernel(int *input, int *result, int *jobs) {
    int job = jobs[threadIdx.x + blockIdx.x * ...];
    int v2 = block_2(block_1(input[job]));
    if (block_3(v1)) result[job] = block_4(v2); }
```

Heap Object Tracer



1. Type inference

- Traverse only classes/methods reachable from main block
- Determine if instance variables are read/written

Actor
-@max_velocity : float / r
-@progress : float / rw
-@street : Street / rw
-@color : Object / (/)

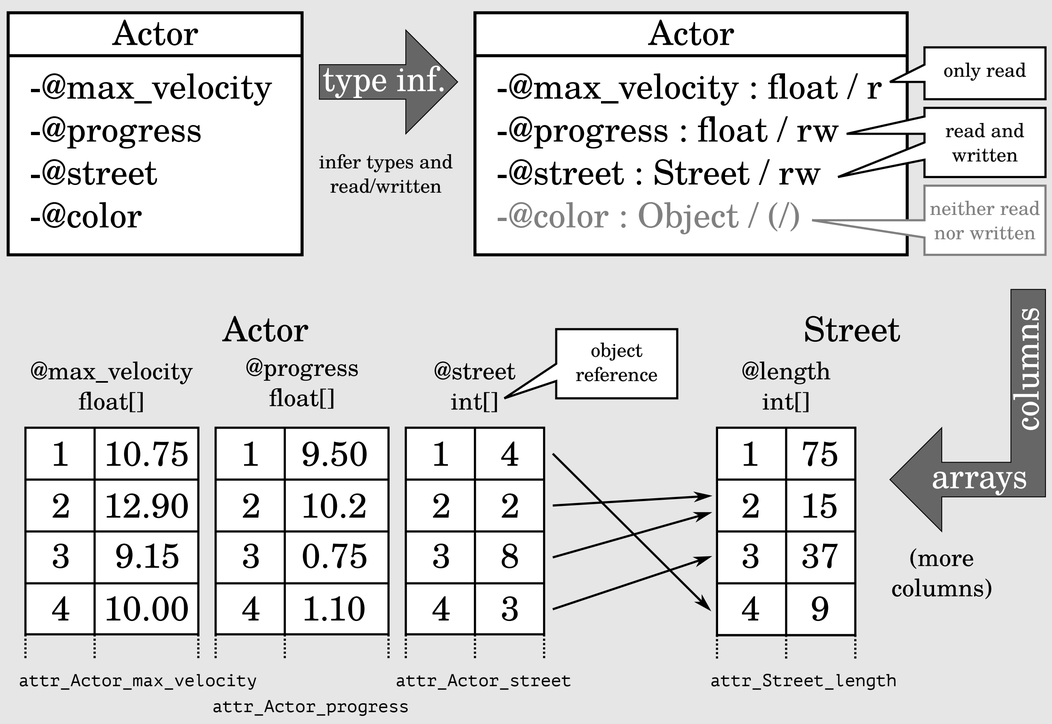
2. Find objects and calc. column offsets

Object graph traversal. Follow instance variable if:
- Instance variable is marked as read/written
- Value type is not primitive and was not processed yet

3. Write object columns

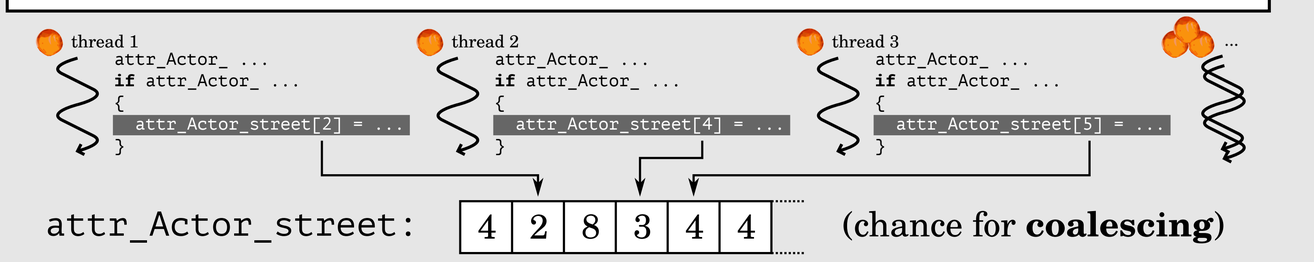
Replace object references with offsets

Columnar Object Layout




```
__device__ void meth_Pedestrian_move(int _id, int weather)
{
   attr_Actor_progress[_id] += meth_Random_rand(attr_Actor_maxProgress[_id]);

   if attr_Actor_progress[_id] >= attr_Street_length[attr_Actor_street[_id]])
   {
      attr_Actor_street[_id] = meth_Street_random_connected(attr_Actor_street[_id]);
   }
}
```

no self for class methods



- 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