Object Support in an
Array-based GPGPU Extension for Ruby

ARRAY ’16

Matthias Springer, Hidehiko Masuhara

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology

June 14, 2016
Overview

Introduction

Example: Agent-based Traffic Simulation

Implementation and Optimizations

Preliminary Benchmarks

Future Work and Summary
What is Ikra?

A Ruby-to-CUDA compiler . . .

- that allows programmers to use GPGPU easily
- with dynamic compilation
- supporting object-oriented programming and polymorphic expressions
- with a number of optimizations: kernel fusion, job reordering, structure-of-arrays data layout
Related Work

• Related work
  − Frameworks similar to Ikra: Accelerate, pyCUDA, …
    Focus on high-level code generation/optimizations (kernel fusion, subexpr. elimination, …)
  − Application-level Optimizations: Programming styles/best practices
    E.g., techniques for reducing branch divergence (e.g., job reordering), data layout optimizations (e.g., structure-of-arrays layout)

• Focus of this work
  − Support object-oriented programming in GPGPU code
  − Employ language-level optimizations to achieve good performance
  − Implement low-level code optimizations
Parallel Sections

- `peach, pmap, pnew, (pselect, preduce)`
- One thread per base array element
- **Input data**: iterator variables, lexical variables, instances variables of objects
- **Output data**: result of parallel section, changed objects (kernel code can have side effects)

```ruby
inc = 10  # lexical variable

[1, 2, 3].pmap do |v|
  v + inc
end
```
Overview

Introduction

Example: Agent-based Traffic Simulation

Implementation and Optimizations

Preliminary Benchmarks

Future Work and Summary
Example: Agent-based Traffic Simulation

Problem Description [2]

- Simulate movement of agents (cars, etc.) on a street network
- Iteration-based, different behavior per *type/class*
Example: Agent-based Traffic Simulation

Problem Description [2]

- Simulate movement of agents (cars, etc.) on a street network
- Iteration-based, different behavior per type/class
Example: Agent-based Traffic Simulation

Problem Description [2]

• Simulate movement of agents (cars, etc.) on a street network
• Iteration-based, different behavior per type/class
Iteration-based Simulation

agents = # load scenario from file system
ticks = 1000
weather = Weather::Rainy

agents.peach(ticks) do |agent|
  agent.move(weather)
end

- One thread per agent
- Syntactical sugar (+synchronization)
Overview

Introduction

Example: Agent-based Traffic Simulation

Implementation and Optimizations

Preliminary Benchmarks

Future Work and Summary
Architecture: Compilation Process

- Code analysis at runtime (dynamic compilation)
- Metaprogramming, reflection allowed outside of parallel sections, but not inside them
- Support for object-oriented programming, Ruby classes, virtual method calls, dynamically-typed expressions
Translation Process

- Block $\rightarrow$ C++ CUDA function
- Instance method $\rightarrow$ C++ CUDA function
- Polymorphic expressions: union type struct [1]

```c
typedef struct union_type
{
    int object_id;
    int class_id;
} union_t;
```
Polymorphic Method Calls

```c
__global__ void kernel(union_t *agent, int weather, int ticks)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    block(agent[tid], weather, ticks);
}

__device__ void block(union_t agent, int weather, int ticks)
{
    for (int i = 0; i <= ticks; i++)
    {
        switch (agent.class_id)  // determined during type inference
        {
            case TAG_Car:
                method_Car_move(agent.id, weather); break;
            case TAG_Pedestrian:
                method_Pedestrian_move(agent.id, weather); break;
        }
    }
}
```
Job Reordering (1/2)

Without Job Reordering:

With Job Reordering:

- **Purpose**: Avoid branch divergence (GPU is SIMD) [6]
- **Mechanism**: Reorder jobs according to runtime type information
- **About 30% faster with job reordering**
__global__
    void kernel(union_t *agent, int *jobs, int weather, int ticks)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    block(agent[jobs[tid]], weather, ticks);
}

__device__ void block(union_t agent, int weather, int ticks)
{
    /* ... */
}
Memory Coalescing

- **Memory Coalescing**: Process multiple global memory access requests in one transaction
- **Requirement**: Spatial locality of memory

Illustration: realizthat GitHub Gist (https://goo.gl/tjPTZr)
Structure-of-Arrays Representation
Overview (c.f. Columnar Objects [3, 4])

Arrays of Structures (AoS):

Structure of Arrays (SoA):

- **Purpose:** Increase memory coalescing
- **Mechanism:** Spatial locality of instance variables (group inst. vars.)
Structure-of-Arrays Representation
Type Inference and Generating Arrays: Object Tracer

1. **Object Tracing**: Generate set of objects reachable from *base array* and *lexical variables* (only those that have `Ikra::Entity` included)
2. **Inst. Var. Type Analysis**: Collect types of all instance variables
3. **Translation**: Infer types and generate CUDA program
4. **SoA Generation**: Generate arrays for Structure-of-Arrays representation
5. **Kernel Invocation**: Run kernel
Kernel Fusion

- **Purpose**: Reduce global memory access for cascaded kernel operations
- **Mechanism**: Generate single kernel for multiple parallel sections [5]

```
__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(v2)) result[job] = block_4(v2); }
```
Kernel Fusion
Examples / Use Cases / Future Work

**Embedded DSL for Database Queries**

```ruby
employees.pselect do |empl|
  e.age > 25
end.preduce([:state]) do |
  acc, empl|
  acc + 1
end

```

**Iteration-based Simulations**

```ruby
agents = # load scenario
for i in 1..ticks
  agents = agents.pmap do |
    agent|
    agent.move
  end
end
```

**Algorithmic Primitives for Graph Algorithms**

```ruby
d_s1 = graph.dist_from(s1)
d_s2 = graph.dist_from(s2)
d_s1.join(d_s2) do |
  n1, n2|
  n1.dist = n2.dist
end
```
Overview

Introduction

Example: Agent-based Traffic Simulation

Implementation and Optimizations

Preliminary Benchmarks

Future Work and Summary
Kernel Running Time

- **without reordering**
- **with reordering**

### Setting:
- nVidia Tesla K20Xm, Ruby 1.9.3p448, Linux 3.0.76-0.11
- Scenario: 4,096 cars, 16,384 pedestrians, 500 streets, 1,000,000 iterations
Job Reordering

Setting: Intel Xeon X5670 CPU (2.93 GHz), Ruby 1.9.3p448, Linux 3.0.76-0.11
Object Tracing and SoA Generation

Setting: Intel Xeon X5670 CPU (2.93 GHz), Ruby 1.9.3p448, Linux 3.0.76-0.11
Overview

Introduction

Example: Agent-based Traffic Simulation

Implementation and Optimizations

Preliminary Benchmarks

Future Work and Summary
Ideas for Future Work

- Full support for *object-oriented programming*: instance creation, etc.
- Job reordering: take into account *run-time types of expressions inside the kernel* (and reorder after a while)
- *Synchronization* primitives: block-level, global
- *Minimizing data transfers*: allocate data only in global memory
- More *low-level optimizations*: e.g., code unrolling for ILP
Summary

- **Ikra**: A GPGPU framework for Ruby
- Supports **object-oriented programming** including virtual method calls and dynamically-typed expressions
- Employs **low-level optimizations**: job reordering, structure-of-arrays data layout, kernel fusion
References


Appendix
Example: Agent-based Traffic Simulation

Graph Representation
Example: Agent-based Traffic Simulation

UML Class Diagram

- Car moves with velocity $\min(S.\text{max\_velocity}, C.\text{max\_velocity})$
- Pedestrian moves with random velocity between -2 mph and 4 mph
- Agent moves to random neighboring street when reaching end of street
Iteration-based Simulation (1/3)

First approach: Parallel inner section

agents = # load scenario from file system
ticks = 1000
weather = Weather::Rainy

for i in 1..ticks
  agents.peach do |agent|
    agent.move(weather)
  end
end

• One thread per agent
• Problem: Separate kernel launches for every iteration
Iteration-based Simulation (2/3)

Second approach: Parallel outer section with loop inversion

```ruby
agents = # load scenario from file system
ticks = 1000
weather = Weather::Rainy

agents.peach do |agent|
  for i in 1..ticks
    agent.move(weather)
    # add synchronization here
  end
end
```

- One thread per agent
Iteration-based Simulation (3/3)

Third approach: Syntactical sugar

agents = # load scenario from file system
ticks = 1000
weather = Weather::Rainy

agents.peach(ticks) do |agent|
  agent.move(weather)
end

• One thread per agent
• Syntactical sugar for previous example (+synchronization)
Job Reordering

- **Purpose:** Avoid branch divergence (GPU is SIMD) [6]
- **Mechanism:** Reorder jobs according to runtime type information
Structure-of-Arrays Representation

Code Example

```c
__device__ float *d_Car_max_velocity;
__device__ float *d_Car_progress;
/* ... */

__device__ void method_Car_move(int agent_id, int weather)
{
    /* ... */

    // Due to SIMD, all threads execute this simultaneously:
    d_Car_progress[agent_id] += speed / 60.0;

    /* ... */
}
```
Structure-of-Arrays Representation

Arrays

- **Basic Idea**: Treat arrays like other classes, but distinguish between inner types
- **Implementation**: Store size and offset into contents array as if they were instance variables
- **Future Work**: Allow arrays to grow
Structure-of-Arrays Representation

SoA Generation (c.f. system tracer in Smalltalk systems)

- Pointers of object references must be replaced with array indices

1. Assign IDs to objects (grouped by class), build hash map object \rightarrow ID
2. Build arrays, replace object references with IDs (or union type tuple)