Iterative Stencil Computations in Ruby on GPUs

Matthias Springer Peter Wauligmann Hidehiko Masuhara

Department of Mathematical and Computing Science, Tokyo Institute of Technology

What is Ikra?

- RubyGem for **array-based GPU computing**
- Compiles Ruby code to C++/CUDA program
- Current focus: Iterative scientific computations
- Parallel `map`, `reduce`, `stencil`, `new`
- Data types inside parallel/host sections: primitive (int, float, bool, nil), array (read only), zipped, object (partial support, incl. method calls), union type (combination of above ones)

Design Decisions

- **Modularity:** Build complex programs from multiple parallel sections using object-oriented programming
- **Kernel Fusion:** Combine parallel sections into single GPU kernel, delay execution to the latest possible point
- **Host Section:** Avoid switching between Ruby interpreter and generated C++ program

Ikra API: Example

```ruby
result = Ikra.host_section do
    arr = Array.pnew(10) do |i| i + 1 end
    while arr.preduce(:+)[0] < 100
        arr = arr.pmap do |i| i + 2 end
    end
    arr
end

puts "Result is #{result.to_a}"
```

(host section)

Symbolic exec. in Ruby interpreter: returns a *command* (contains all information for code generation + execution)

```ruby
a = Array.pnew(dimensions: [10, 6]) do |idx|
    idx[0] + idx[1] + idx[2]
end

b = a.pstencil([[-1, 1], [0, 0]], 0) do |v, i|
    value_sum = v[-1, 1] + v[0, 0]
    index_sum = i[0] + i[1]
    value_sum + index_sum
end
```

multi-dim. array

out of bounds value

neighborhood (relative indices)

Kernel Fusion in Loops via Symb. Execution

```
a1 = Arr.pnew(...)                         -- C_new[int]
                                           -- a1 ⊕ a4 = C_new[int] ⊕ a4
while (a2 = ϕ(a1, a4); a2.preduce[0] < 100)
    a3 = a2.pmap do ... end                -- C_map[C_new[int] ⊕ a4]
    a4 = a3.pmap do ... end
end                                        -- C_map[C_map[C_new[int] ⊕ a4]]  (= a4)

return a2
```

$a_1 \oplus a_4$

$= C_{new}[int] \oplus C_{map}[C_{map}[C_{new}[int] \oplus \bigstar]]$ ($= \bigstar$)

$= a_1$ (no iteration)

$\oplus\ C_{map}[C_{map}[C_{new}[int]]$ (after 1 iteration)

$\oplus\ C_{map}[C_{map}[C_{map}[C_{map}[C_{new}[int]]]]$ (after 2 iterations)

$\oplus\ \ldots$ (after more iterations)

Type/command objects guide code generation. Cannot generate code for infinitely large types!

Type Inference on Host Section AST in SSA Form.
The type of a parallel section is the result of its evaluation in the Ruby interpreter.

AST Rewriting:
Launch kernel in loop explicitly to break the cycle.

```
a1 = Arr.pnew(...)

while (a2 = ϕ(a1, a4); a2.preduce[0] < 100)
    a3 = (a2.run).pmap do ... end          -- Array[int]   C_map[Array[int]]

    a4 = a3.pmap do ... end                -- C_map[C_map[Array[int]]]
end

return a2                                  -- a1 ⊕ a4 = C_new[int] ⊕ C_map[C_map[Array[int]]]
```

```
while ((Arr.pnew ⊕ a2.pmap.pmap).preduce[0] < 100)
    a2 = Arr.pnew ⊕ a2.pmap.pmap
end

return Arr.pnew ⊕ a2.pmap.pmap
```

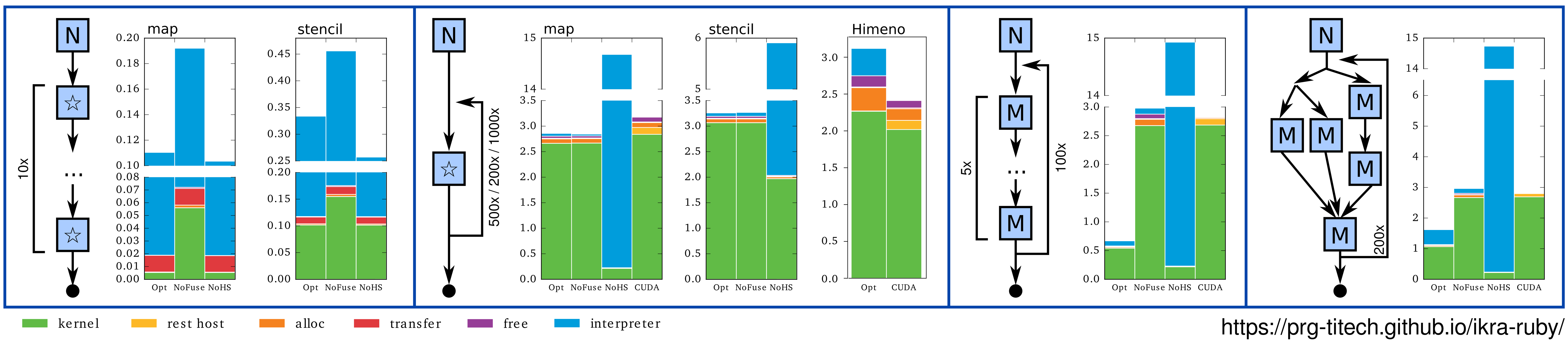
Code Generation:
High-level overview with kernel launches only

Code Generation

- C++ type for polymorphic expressions: union type struct

```cpp
struct union_t {
    union { int int_; /* ... */ void *pointer; } data;
    int class_id;
}
```

- Method call with polymorphic receivers: switch stmt.
- Parallel section: Data structure for command data
- Kernel launch: Generated only for `run`, `[]`, end of section
- Future work: Data sharing between multiple parallel sections (avoid redundant comput.), escape analysis to detect if it is safe to reuse the same memory location



https://prg-titech.github.io/ikra-ruby/