Julia 编程语言探索

小组成员：蔡文韬、戴路、董恒、何纪言、何理扬、俞晨东、张立夫

指导老师：张昱

Outline

0x00: 项目介绍

0x01: Julia 语言基础

0x02: Julia 语言的应用

0x03: 收获总结和团队分工

0x00: 项目介绍

1. 新兴应用领域带来的新挑战
2. Julia 介绍

新兴应用领域带来的新挑战

近年来，越来越多的**新兴应用领域**给编程语言带来了全新的挑战：

机器学习

数据科学

异构计算

……

新兴应用领域带来的新挑战 (con't)

以**机器学习**为例：

数值计算（矩阵和其他代数运算）

微分计算（Automatic Differentiation 自动微分）

算法设计（计算图模型，可微分编程，算法框架）

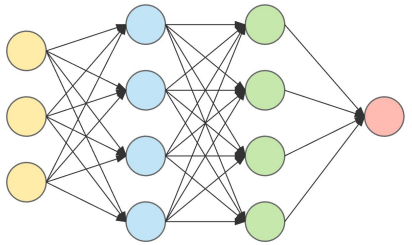
并行计算（异步模型，通信模型，分布式计算）

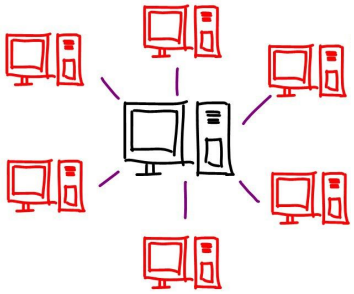
计算效率（编译优化，运行优化，JIT 技术）

异构计算（GPU 编程，TPU 编程，FPGA 编程）

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} = \begin{bmatrix}4 & 4\\10 & 8\end{bmatrix}$$

$$\begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} \times \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}2 & 4\\7 & 10\end{bmatrix}$$





新兴应用领域带来的新挑战 (con't)

“金无足赤，人无完人”，传统编程语言由于其产生背景或其他原因在应对新兴应用带来的挑战时不一定能完美胜任。

Julia 作为一种被精心设计的现代编程语言，在很多方面相较传统编程语言，可能更加适合解决这一类问题。我们的探索调研也将从这些方面展开：

- **语言特性层面；**
- **编译运行机制层面；**
- **语言扩展和库层面；**

Julia 介绍

Julia 诞生于“**贪心**”（**来自 *Why We Created Julia***）：

- **Open Source 自由开源；**
- **Fast 快速；**
- **General 通用；**
- **Dynamic 动态；**
- **Technical 擅长计算；**
- **Optionally Typed 可选类型标注；**
- **Composable 可组合；**

Julia 项目起源：MIT

Julia 项目开始时间：2009 年

Julia 1.0 发布：2018 年 8 月

0x01: Julia 语言基础

一、Julia 语言特性

二、Julia 编译和运行机制

三、Julia 性能测试

Julia 语言特性--类型

作为一个科学计算语言，Julia必须拥有丰富的**类型**系统

```
abstract type Number end

abstract type Real <: Number end

primitive type Int64 <: Signed 64 end

struct Polar{T<:Real} <: Number
    r::T
    t::T
end
```

Julia 语言特性--元编程

基本概念

```
julia> ex1 = :(1+2*3)
:(1 + 2 * 3)

julia> dump(ex1)
Expr
  head: Symbol call
  args: Array{Any}((3,))
    1: Symbol +
    2: Int64 1
    3: Expr
      head: Symbol call
      args: Array{Any}((3,))
        1: Symbol *
        2: Int64 2
        3: Int64 3
```

Julia 语言特性--元编程

宏

```
julia> macro sayhello()
           return :( println("Hello, world!") )
       end
@sayhello (macro with 1 method)
```

Julia 语言特性--元编程

强大的**反射**功能用来探索程序的内部结构

```
julia> struct Point
           x::Int
           y
       end

julia> fieldnames(Point)
(:x, :y)
```

Julia 语言特性--多分派

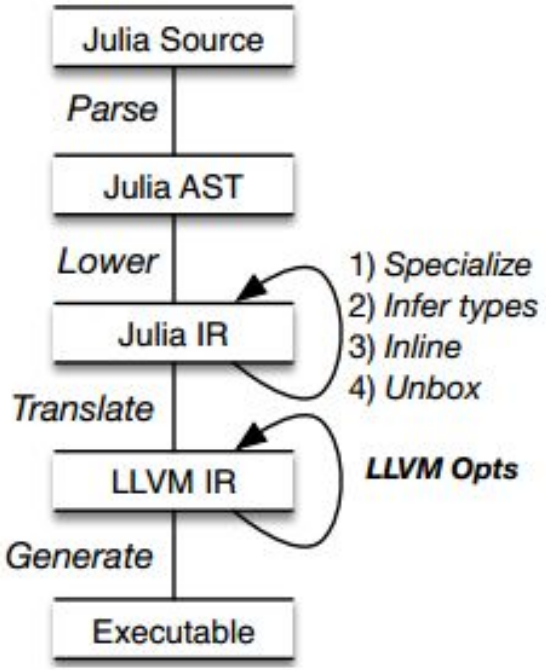
多分派在多态方面起到很大的作用

```
struct Dual{T}
    re::T
    dx::T
end

function Base.:(+)(a::Dual{T},b::Dual{T}) where T
    Dual{T}(a.re+b.re, a.dx+b.dx)
end
function Base.:(*)(a::Dual{T},b::Dual{T}) where T
    Dual{T}(a.re*b.re, a.dx*b.re+b.dx*a.re)
end
function Base.:(/)(a::Dual{T},b::Dual{T}) where T
    Dual{T}(a.re/b.re, (a.dx*b.re-a.re*b.dx)/(b.re*b.re))
end
```

Julia 编译和运行机制

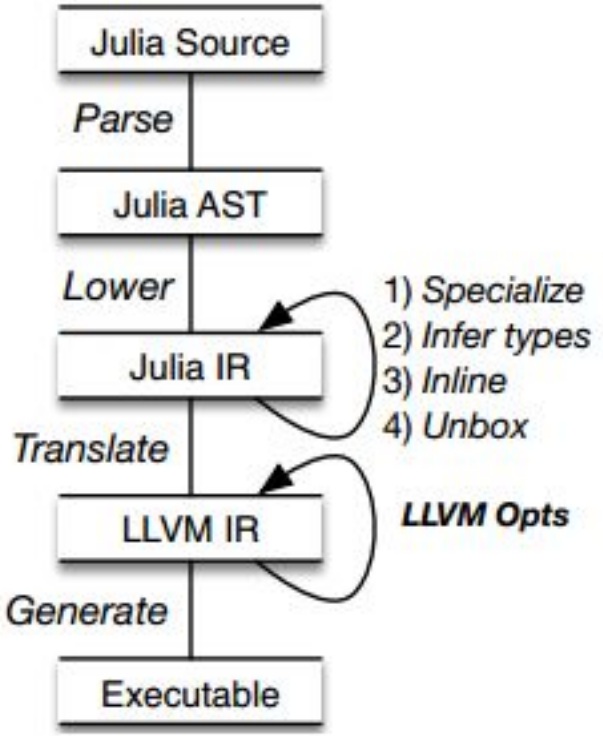
1. 从源代码生成抽象语法树
2. 从抽象语法树得到Julia IR
3. 对Julia语言的中间表示进行优化
4. 生成LLVM的中间表示
5. 在运行过程中生成可执行代码



Julia 编译和运行机制

Julia IR层面的四项重要优化：

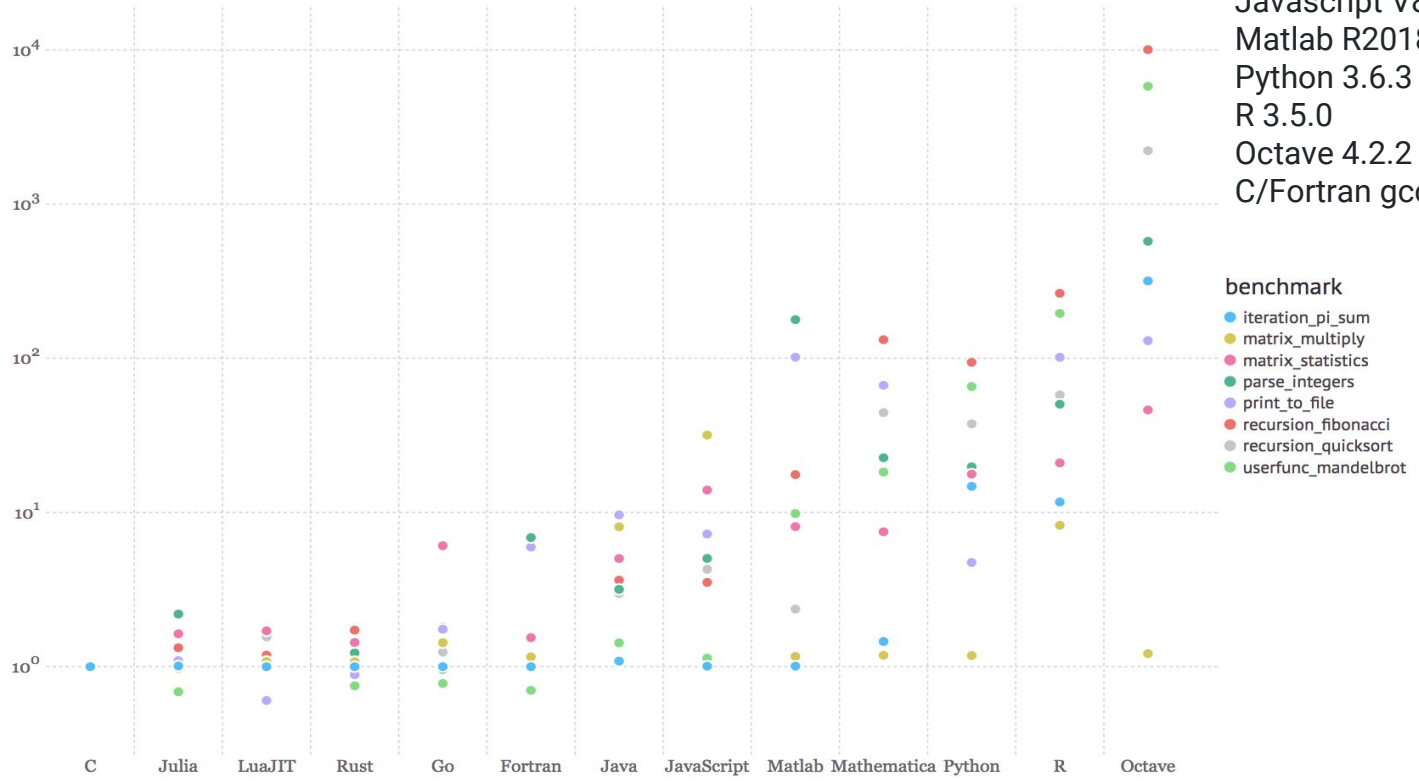
1) Method Specialization(方法特化)
2) Type Inference(类型推断)
3) Method Inlining(内联)
4) Object Unboxing(拆箱)



Julia 性能测试

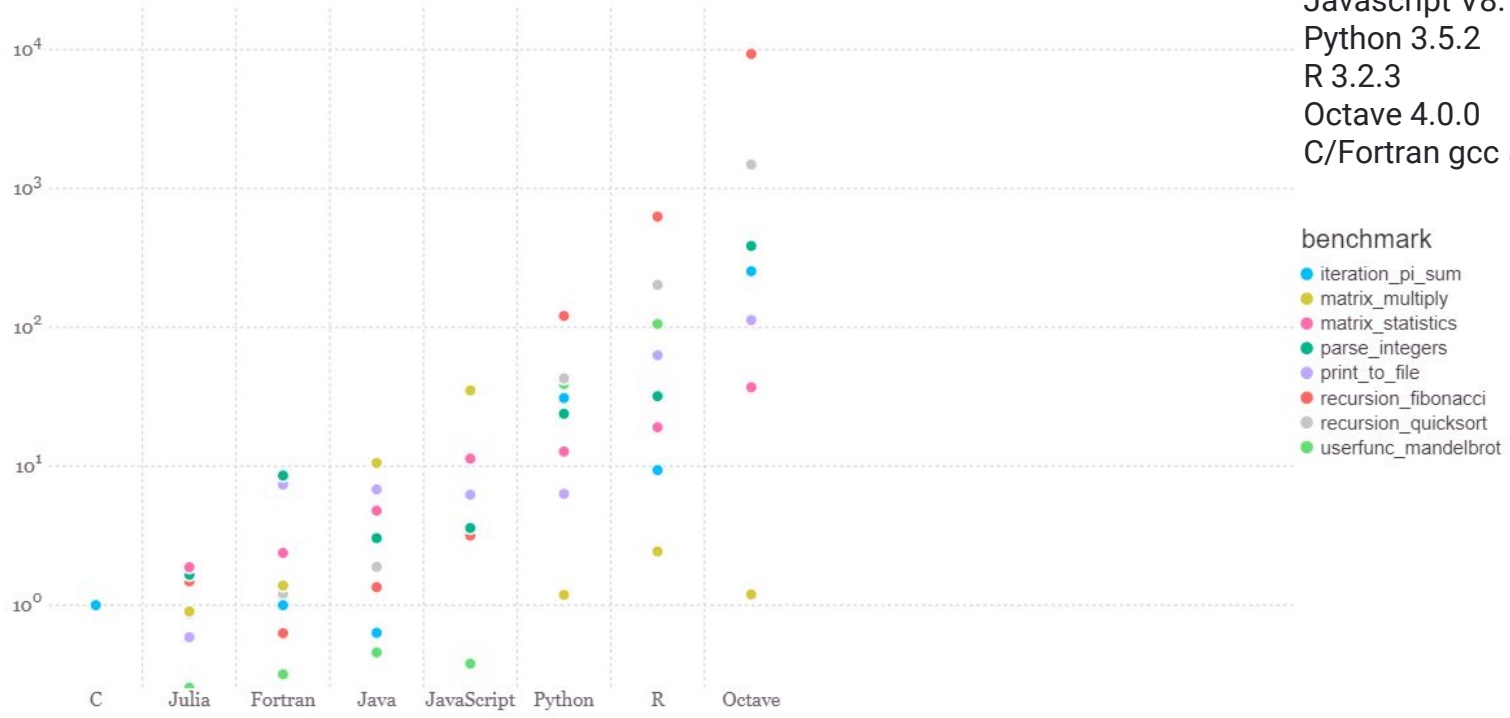
官方的测试结果:

Julia v1.0.0
SciLua v1.0.0
Rust 1.27.0
Go 1.9
Java 1.8.0
Javascript V8.6.2
Matlab R2018a
Python 3.6.3
R 3.5.0
Octave 4.2.2
C/Fortran gcc 7.3.1



Julia 性能测试

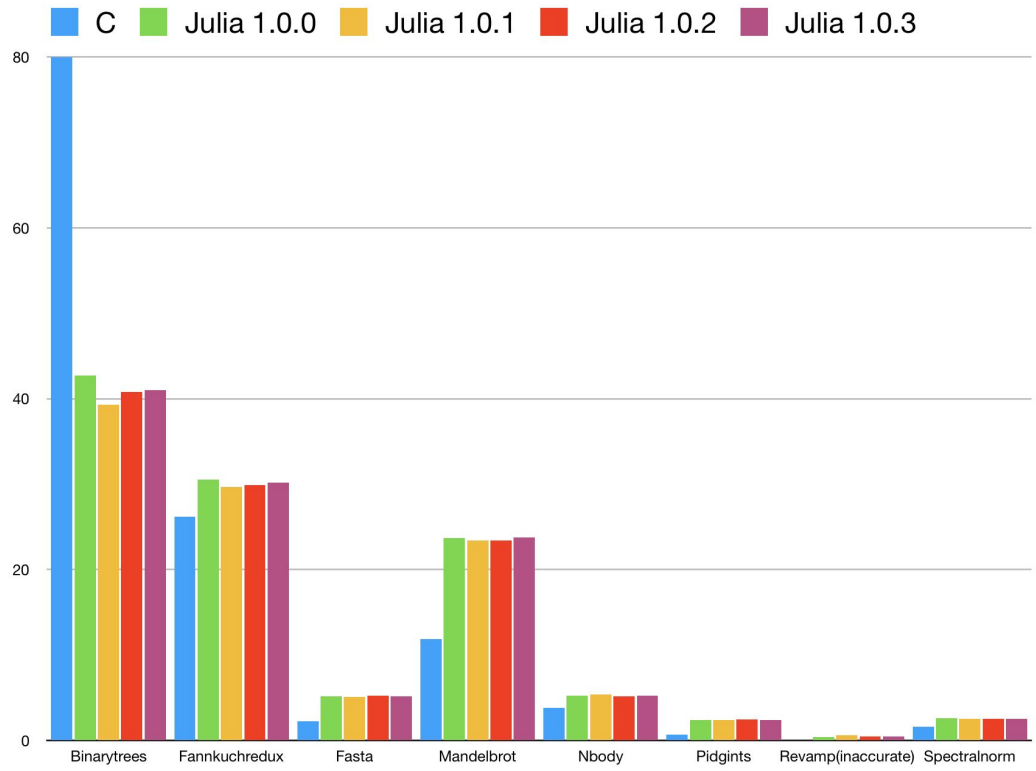
我们的测试结果:



Julia v1.0.2
Java 1.8.0
Javascript V8.15.0
Python 3.5.2
R 3.2.3
Octave 4.0.0
C/Fortran gcc 5.4.0

Julia 性能测试

我们根据PLBG中的测试方法与测试样例得到的测试数据:



0x02: Julia 语言的应用

一、Julia 与并行计算

二、Julia 与强化学习

三、Julia 与异构计算

Julia 与并行计算

Julia 语言提供了不同层次的并行状态：

1. **Julia Coroutines** (Green Threading)

   标准库中对**计算任务**的抽象

2. **Multi-Threading**

   语言层面原生支持**多线程**

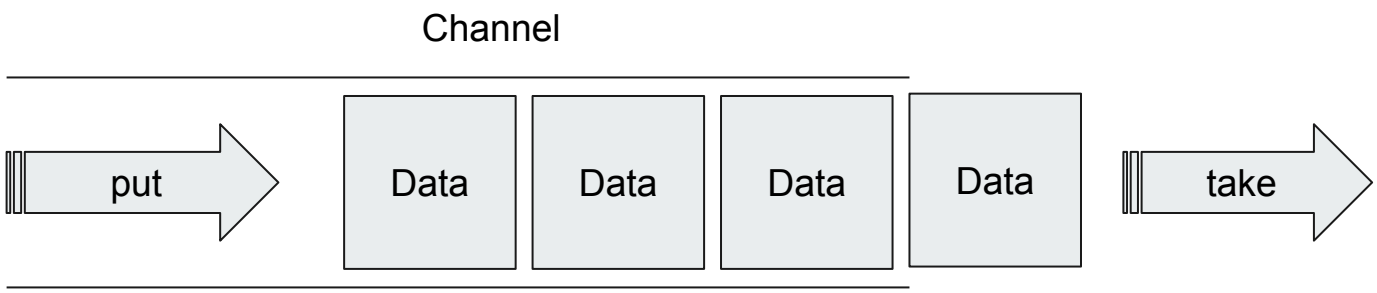
3. **Multi-Core or Distributed Processing**

   多核心/多机器分布式计算

Julia 与并行计算：Coroutines

Channel

- 类似的概念：FIFO 队列
- 和 Task 配合使用，解决类似“生产者-消费者”问题；



Julia 与并行计算：Coroutines

Task

- 类似的概念：协程，轻量级线程，……；
- 可以是：
  - 一个**计算任务**；
  - 一个**消费者**；
  - 一个**生产者**；
- 特点：
  - 切换 Task 并不使用任何空间；
  - Task 可以以任意次序切换

Julia 与并行计算：Multi-Threading (Experimental)

Threading

Julia 在语法上支持多线程：

- 底层实现在不同操作系统上不同；
- 提供易用的标准库语法和宏；
- 提供**原子操作**来避免部分竞争；
- 提供**同步机制**来避免其他竞争；

Julia 与并行计算：Multi-Core or Distributed Processing

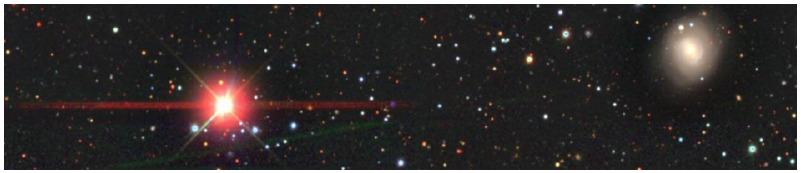
Distributed

- 类似的机制：MPI
  - 不同之处在于 Julia 中的通信通常是单向的；
- 提供方便使用的接口和宏；
- 支持**本地集群**和**跨机器集群**；

```
using SharedArrays

a = SharedArray{Float64}(10)
@distributed for i = 1:10
    a[i] = i
end
```

Julia 并行计算的应用



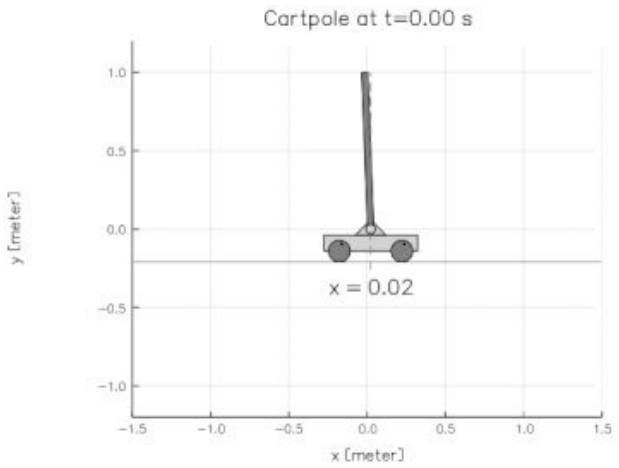
Celeste.jl：在天文图像中发现并描绘恒星与星系

该库测试时使用了 650,000 核，计算速度达到了 1.5 PetaFLOP/s

并行计算仍有较大发展空间：

1. 多线程的实现与调度算法较为基础，可进一步优化；
2. 目前对 Julia 并行计算的使用并不广泛，如 Flux.jl 和 Zygote.jl 都没有直接使用到并行计算的特性；

Julia与强化学习：Overview



Cartpole

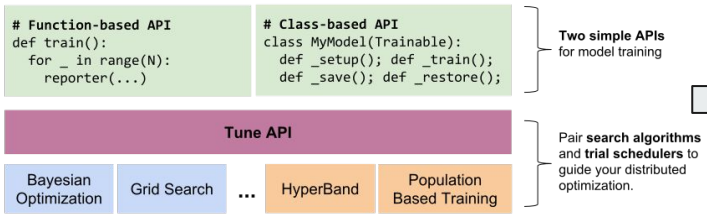
什么是RL?

Env, Policy, Reward

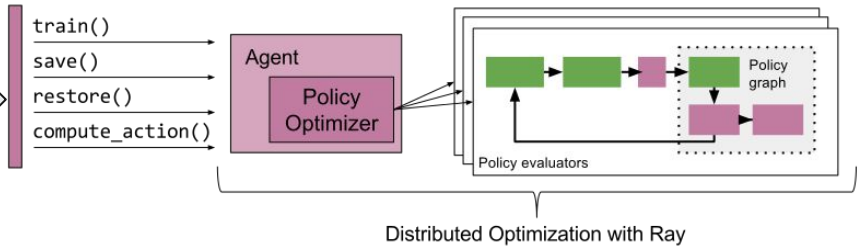
1. Julia对RL的支持度如何？Toy Implements!
   a. 相关库数量少且不完善：JuliaML
   b. 封装其他语言的库 ：OpenAIGym
   c. 架构薄弱：分布式
2. 借鉴其他语言的库
   a. 分布式计算框架：Ray
   b. ML超参搜索：Tune
   c. 可拓展RL框架：RLlib
   d. 更高级的抽象：RLgraph
3. 直接从RL算法抽象框架特征

Julia与强化学习：Ray系列，Why Ray?

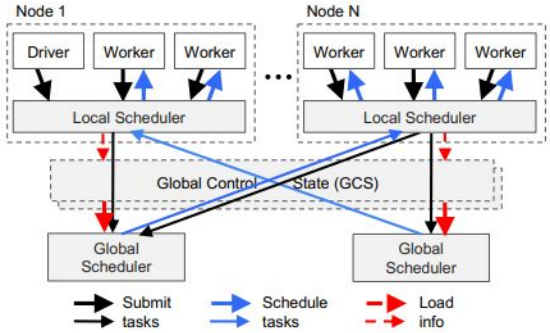
2. 深度学习超参搜索：Tune



3. 可拓展RL框架：RLlib



1.分布式执行：Ray底层



4. And More...

- 多种算法实现：A3C/DQN/PPO...
- 多种优化策略：Allreduce/MultiGPU/Asnchronous...
- 多种环境实现：OpenAI/Custom/...
- 抽象高级、扩展简单

Julia与强化学习：RLgraph & RL 算法

RLlib缺点：

1. 与深度学习框架耦合
   TensorFlow/PyTorch/Keras
2. 数据流/控制流纠缠
   代码重用/子模块测试

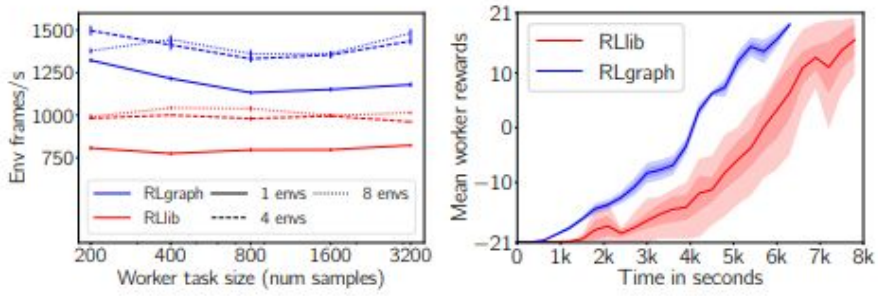
RL Workloads：

1. 采样数据高效管理
   Policy传递/采样传递
2. 多样化资源/时间需求
   CPU/GPU millisecond/hours
3. 与环境频繁的交互
   数据逐步产生/动态执行
4. 高效开发与测试
   模块分离

RLgraph:

1. 更高层抽象：Component组块
2. 不依赖具体DL框架

But... 开发未完善



(a) Single worker throughput. (b) Training times for *Pong*.

使用Julia给机器学习造轮子

- *Doing the obvious thing.*
- *You could have written Flux.*
- *Play nicely with others*

Julia的设计目的在于保持C语言的计算速度，同时保持高级语言的方便性

基于这样的思想，flux对机器学习进行了更高层面的抽象，将机器学习的算法和基本操作等封装为函数。

相比于sklearn等基于python的封装库，flux可以充分利用julia在数值计算，并行计算的速度优势。

使用Julia实现自动微分：Julia编译与计算图的结合

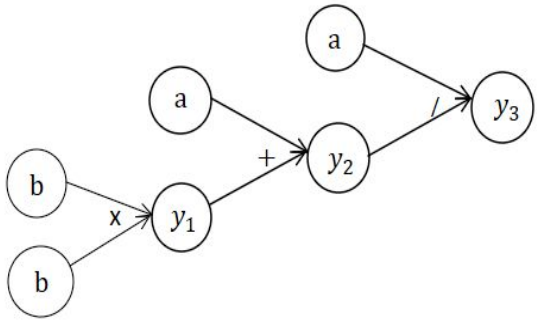
Tensorflow等深度学习框架均基于计算图模型。

例如，计算 $y = f(a,b) = \dfrac{a}{a+b^2}$

可以分为以下几个步骤：

$$
\begin{aligned}
y_1 &= b^2 \\
y_2 &= a + y_1 \\
y_3 &= \frac{a}{y_2}
\end{aligned}
$$

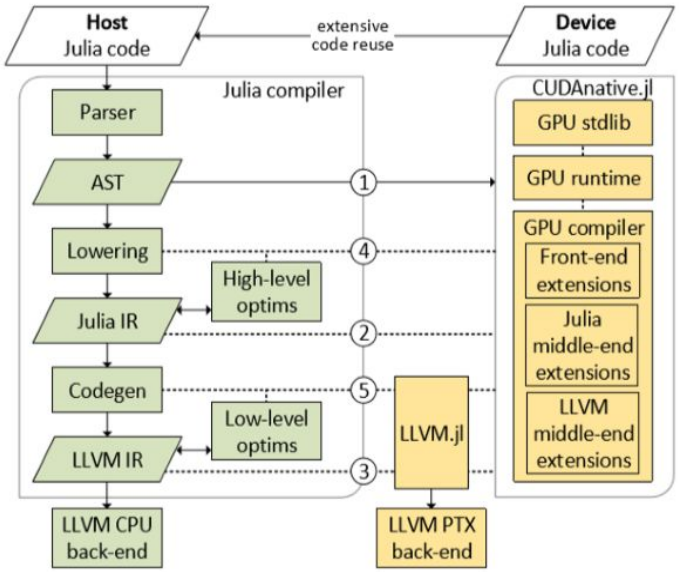
：



在计算图模型中可以利用链式法则逆向求微分

由于(左边)计算的流程类似于编译中的SSA，因此zygote的设计者通过在编译器的层面中进行处理，使得程序能直接通过SSA IR计算微分，避免构建计算图

Julia对GPU编程的支持



Julia代码与GPU代码之间的交互关系

传统方案(比如python)支持GPU编程的方法为:

生成**通用IR(LLVM)**, 再由通用IR从头生成可以生成GPU代码的IR。该方法的代码的复用性, 功能性都较差。

Julia提出的方法为:在AST, 高级IR中均提供和GPU的交互, 该方法的代码复用性更好;且能够尽早对代码作优化。

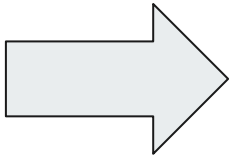
0x03: 收获总结和团队分工

一、Julia

二、团队合作

Julia 特点总结

- 多重派发
- 擅长数据计算
- 同时兼具各种语言特性
- 速度接近 C 系语言
- 元编程
- 封装
- 用户友善
- ……



- 并行计算
- 机器学习
- 异构计算

Julia 语言以及生态存在的问题

- **运算性能**并非绝对优势；
- **启动和编译延迟**时间长；
- 特性支持**太多太杂**，没有针对应用场景优化；
- 与其他语言的**互相调用**问题（相比于 Python/Java 等，Julia 没有很好地支持外部调用）；
- **常用工具包**缺失；
- 各种包的**稳定性**仍然不够好；

更多可以改进之处

- 针对应用场景，充分利用 Julia 语言特性，如：
  - **自动微分和可微分编程**；
  - 更灵活和更复杂的**编译时/运行时优化**；
- 完善机器学习相关工具包，算法框架的功能性，易用性，增加机器学习应用的**开发效率**；
- 充分挖掘 Julia 在**并行计算**和**异构计算**上的优势，增加应用的**运行效率**；

团队分工

- 俞晨东：Julia语言特性，编译优化，性能测试
- 董恒：强化学习JuliaML，Ray框架
- 张立夫：并行计算，Ray框架
- 戴路：机器学习库调研，Ray框架
- 何理扬：Julia语言特性，编译过程，性能测试
- 蔡文韬：Julia语言特性，编译过程
- 何纪言：Julia数值计算，多派发，并行，源码分析性能测试

团队合作的启发与反思

- 成员之间的沟通：任务分配，及时汇总交流
- 明确需求

谢谢！

欢迎提出问题