Научное программирование на языке Julia

Камиль Хайруллин

ООО «РИТМ»

Что такое Julia

- Открытый динамически компилируемый язык программирования, ориентированный на высокопроизводительные научно-технические вычисления.
  https://julialang.org/
- Разработан в MIT. Развивается в лаборатории Julia Lab при поддержке Julia Computing.
- Open Source, MIT License.
- Разрабатывается с 2009 г.
  Версия 1.0.0 – 2012 г.
  Текущая версия – 1.8.5.

Особенности Julia

- Динамическая типизация как в Python и MATLAB с возможностью явного указания типа.
- Высокая производительность и кроссплатформенность за счёт компиляции для LLWM.
- Множественная диспетчеризация.
- Встроенные в язык средства параллельного исполнения кода и управления вычислительным кластером.
- Воспроизводимые рабочие окружения.
- Полноценный язык общего назначения с возможностью асинхронного и метапрограммирования.
- Современная экосистема разработки: встроенный пакетный менеджер, средства отладки, логгирования и профилирования кода.

Средства разработки Julia

VS Code

VS Code Extension

Jupyter

Jupyter kernel

Pluto.jl

Simple reactive notebooks

Emacs

Emacs plugin

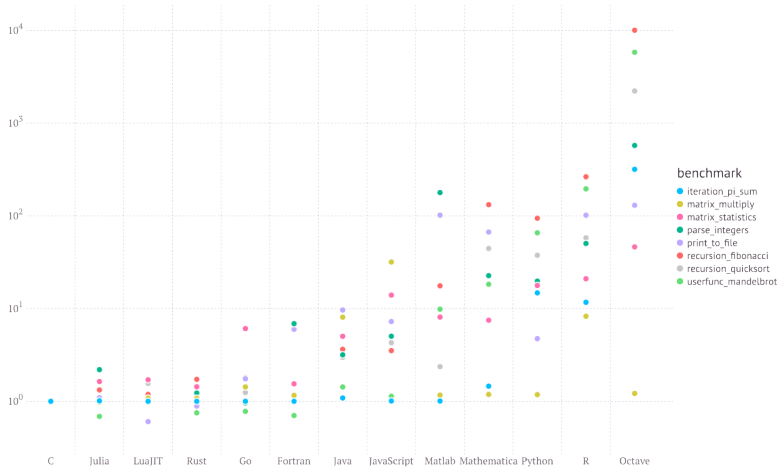
NotePad++

Notepad++

Vim

Vim plugin

Производительность вычислений



https://julialang.org/benchmarks/

Режижмы REPL - Read-Eval-Print-Loop

Активатор	prompt	Назначение
по умолчанию	`julia>`	Основной режим. Вычисление выражений.
;	`shell>`	Режим командной строки ОС.
?	`help>`	Получение справки по модулю, сруктуре или функции.
]	`pkg>`	Пакетный менеджер. Установка и удаление пакетов. Работа с окружениями.

Сравнение синтаксиса

Julia

```julia
# Julia comment

#=
    Multiline comment
=#

if i <= N   # condition
    # actions
else
    # alternate actions
end

for i = 1:N
    # actions
end

function func(x::Float64)::Float64
    x^2
end
```

MATLAB

```matlab
% MATLAB comment

%{
    Multiline comment
}%

if i <= N   % condition
    % actions
else
    % alternate actions
end

for i = 1:N
    % actions
end

function func(x)
    x^2
end
```

Python

```python
# Python comment

#
# Multiline comment
#

if i <= N:  # condition
    # actions
else:
    # alternate actions


for i in range(N):
    # actions


def func(x):
    x^2
```

Обзор синтаксиса

- **Переменные**.
  Поддержка UTF-8: ж, σ, $\sum$, a_1, и т.д.
- **Функции**.

```
sum(a, b) = a + b
sum = (a, b) -> a + b
```

- **Конвейер и композиция**.

```
sin(cos(tan(π/6)))

π/6 |> tan |> cos |> sin

(sin ∘ cos ∘ tan)(π/6)
```

- **Структуры**.

```
struct Point
    x
    y
    z
end
```

```
mutable struct Point
    x::Float64
    y::Float64
    z::Float64
end
```

- **Параметрические типы**.

```
struct Point{T}
    x::T
    y::T
    z::T
end
```

Обзор синтаксиса

- **Множественная диспетчеризация.**

```
f(x::Int64) = x^2

f(x::Float64) = 2x
```

```
norm(p::Point) = sqrt(p.x^2 + p.y^2 + p.z^2)

function norm(p::Point{T}) where {T <: Integer}
    p.x^2 + p.y^2 + p.z^2 |> sqrt |> floor |> T
end
```

- **Макросы**

```
macro my_macro(expr) ... end

reactions = @reaction_network begin
    c₁, S + E --> SE
    c₂, SE --> S + E
    c₃, SE --> P + E
end c₁ c₂ c₃
```

- **Модули.**

```
module MyModule
    using DifferentialEquations

    export f

    f(x, y) = x + g(y)
    g(x) = 2x
end # module MyModule
```

Пример: вычисление СКО

Эталонная реализация на C

```c
#include <stdlib.h>
#include <math.h>

double mean(double *vec, size_t n) {
  double sum = 0.0;
  for (size_t i = 0; i < n; i++) {
    sum += vec[i];
  }
  return sum / n;
}

double stddev(double *vec, size_t n) {
  double avg = mean(vec, n);
  double dispersion = 0.0;
  for (size_t i = 0; i < n; i++) {
    double diff = vec[i] - avg;
    dispersion += diff * diff;
  }
  return sqrt(dispersion / (n-1));
}
```

Результаты замера производительности на массиве из 1M элементов, 3.5K запусков.

Без векторных операций:
Минимальное время: 1.372 мс
Максимальное время: 1.687 мс

С векторными операциями `-ffast-math -mavx`:
Минимальное время: 0.343 мс
Максимальное время: 0.781 мс

Пример: вычисление СКО

Релизация на Python.

```python
from math import sqrt
import numpy as np

def mean(vec):
  sum = 0.0
  for x in vec:
    sum += x
  return sum / len(vec)

def stddev(vec):
  avg = mean(vec)
  dispersion = 0.0
  for x in vec:
    diff = x - avg
    dispersion += diff * diff
  return sqrt(dispersion / (len(vec) - 1))

def stddev_np(vec):
  n = len(vec)
  avg = np.sum(vec) / n
  diff = vec - avg
  dispersion2 = np.dot(diff, diff)
  return sqrt(dispersion2 / (n-1))
```

Результаты замера производительности на массиве из 1M элементов, 3.5K запусков.

«Чистый» Python:
Минимальное время: 62.56 мс
Максимальное время: 76.32 мс

NumPy:
Минимальное время: 7.57 мс
Максимальное время: 10.15 мс

Пример: вычисление СКО

Релизация на Julia.

```
@fastmath function mean(vec)
  sum = 0.0
  for x in vec
    sum += x
  end
  return sum / length(vec)
end

@fastmath function stddev(vec)
  avg = mean(vec)
  dispersion = 0.0
  for x in vec
    diff = x - avg
    dispersion += diff * diff
  end
  return sqrt(dispersion / (length(vec) - 1))
end
```

Результаты замера производительности на массиве из 1M элементов, 3.5K запусков.

Без `@fastmath`:
Минимальное время: 1.431 мс
Максимальное время: 2.346 мс

С `@fastmath`:
Минимальное время: 0.162 мс
Максимальное время: 0.950 мс

Прикладные библиотеки

- **Чтение/запись данных**. `DataFrames.jl`, `CSV.jl`.
- **Визуализация данных**. `Plots.jl`, `Makie.jl`.
- **Линейная алгебра**. `LinearAlgebra.jl`.
- **Оптимизация**. `Optim.jl`, `JuMP.jl`.
- **Графы**. `Graphs.jl`.
- **Дифф. уравнения**. `DifferentialEquations.jl`.
- **Символьно-численное моделирование**. `ModelingToolkit.jl`.
- **Метод конечного элемента**. `JuliaFEM.jl`.
- **Нейронный сети**. `Flux.jl`.
- **Химические реакции**. `Catalyst.jl`.

Спасибо за внимание!