ORES
Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК - 004.432.2

ЯЗЫК НАУЧНЫХ ВЫЧИСЛЕНИЙ JULIA

Зырянов Дмитрий Михайлович
Магистрант 1-го курса
НИТУ «МИСиС», кафедра ТМН, Москва, Россия
ZyryanovDmM@yandex.ru

Емельянов Антон Алексеевич
Магистрант 2-го курса
НИТУ «МИСиС», кафедра ТМН, Москва, Россия
Ant.anton.e@yandex.ru

Лещенко Кирилл Сергеевич
Магистрант 1-го курса
МГТУ им. Н.Э. Баумана, кафедра ИУ4 Москва, Россия
kirillbmst@ya.ru

Аннотация

В статье выделена проблема языков программирования для научных вычислений. Читателю предлагается ознакомиться с новым языком программирования Julia, созданным решить эту проблему. Рассмотрены проблемы современных инструментов вычисления, выявлены их сильные и слабые качества. Сделано сравнение производительности языка Julia с другими. Представлена проблематика динамической и статической типизации, дан ответ языка Julia. Рассмотрена реализация параллельных вычислений. Сравнен синтаксис языка Julia и MATLAB. Перечислены особенности языка. Сделаны выводы.

Ключевые слова: язык программирования, Julia, Julialang, MIT, распределённые вычисления, параллелизм, научные вычисления, типизация, производительность, MATLAB, синтаксис, open source.

SCIENTIFIC COMPUTING LANGUAGE JULIA

Dmitry M. Zyryanov
Master student, 1-st year
National University of Science and Technology "MISIS", Moscow, Russia
ZyryanovDmM@yandex.ru

Anton A. Yemelyanov
Master student, 2-st year
National University of Science and Technology "MISIS", Moscow, Russia
Ant.anton.e@yandex.ru

Kirill S. Leshchenko
Master student, 1-st year
Bauman Moscow State Technical University, Moscow, Russia
kirillbmst@ya.ru

ABSTRACT

The article highlights the problem of programming languages for scientific computing. The reader is invited to familiarize themselves with the new programming language Julia, created to solve this problem. The problems of modern computing tools are considered, their strong and weak qualities are revealed. The performance of the Julia language is compared with other languages. The problems of dynamic and static typing are presented, and the answer of the Julia language is given. The implementation of parallel computing is considered. The syntax of the Julia and MATLAB languages is compared. The language features are listed. Conclusions are drawn.

Keywords: programming language, Julia, Julialang, MIT, distributed computing, parallelism, scientific computing, typing, performance, MATLAB, syntax, open source.

Введение

В настоящее время в области научных вычислений очень популярны динамические языки программирования. Они, как правило, являются удобными и мощными, но им не хватает производительности [2].

Статические же языки обладают высокой производительностью, но, чтобы воспользоваться ей, от пользователя требуется высокая квалификация, которую тяжело требовать у непрограммиста [2].

Эта проблема была давно известна в научном сообществе и с развитием науки, она всё больше обострялась.

Цель создания языка

Разработка языка программирования Julia началась в 2009-ом году в Массачусетском технологическом университете. Он заведомо разрабатывался как язык для научных вычислений. Его разработчики пришли в проект, имея большой опыт программирования на Lisp, Ruby, Python, R и MATLAB. В середине 2012 года была выпущена публичная версия. Бета-версия имела много проблем и интересовала только энтузиастов [11].

Вот что пишут создатели языка:

«Мы хотим язык программирования с открытым исходным кодом, с открытой лицензией. Мы хотим скорость C с динамизмом Ruby. Нам нужен гомоиконичный язык с настоящими макросами, как Lisp, но с очевидными знакомыми математическими обозначениями, такими как в Matlab. Мы хотим что-то такое же удобное для общего программирования, как Python, такое же простое для статистики, как R, такое же естественное для обработки строк, как Perl, такое же мощное для линейной алгебры, как Matlab, и способное склеивать программы вместе как оболочку. Нечто простое в освоении, но при этом радующее самых серьезных хакеров. Мы хотим высокой интерактивности и эффективной компиляции. Мы ведь не слишком многого просим, верно?»

На момент написания статьи вышла версия 1.5.2. Язык начали преподавать в университетах США, появились онлайн курсы, языком заинтересовались менеджеры коммерческих проектов, а владение этим языком начало высоко оплачиваться.

Рассмотрим, чего добились создатели.

Производительность

Одним из главных качеств, которым должен обладать язык программирования для научных вычислений — это производительность [9]. Эталоном на этом поприще является язык C. Создатели языка Julia смогли приблизиться к скорости языка программирования C, а в некоторых случая даже превзошли ее.

Чтобы достаточно полно оценить производительность языка, необходимо провести ряд тестов сравнения с другими языками программирования. Данное сравнение представлено на рисунке 1.

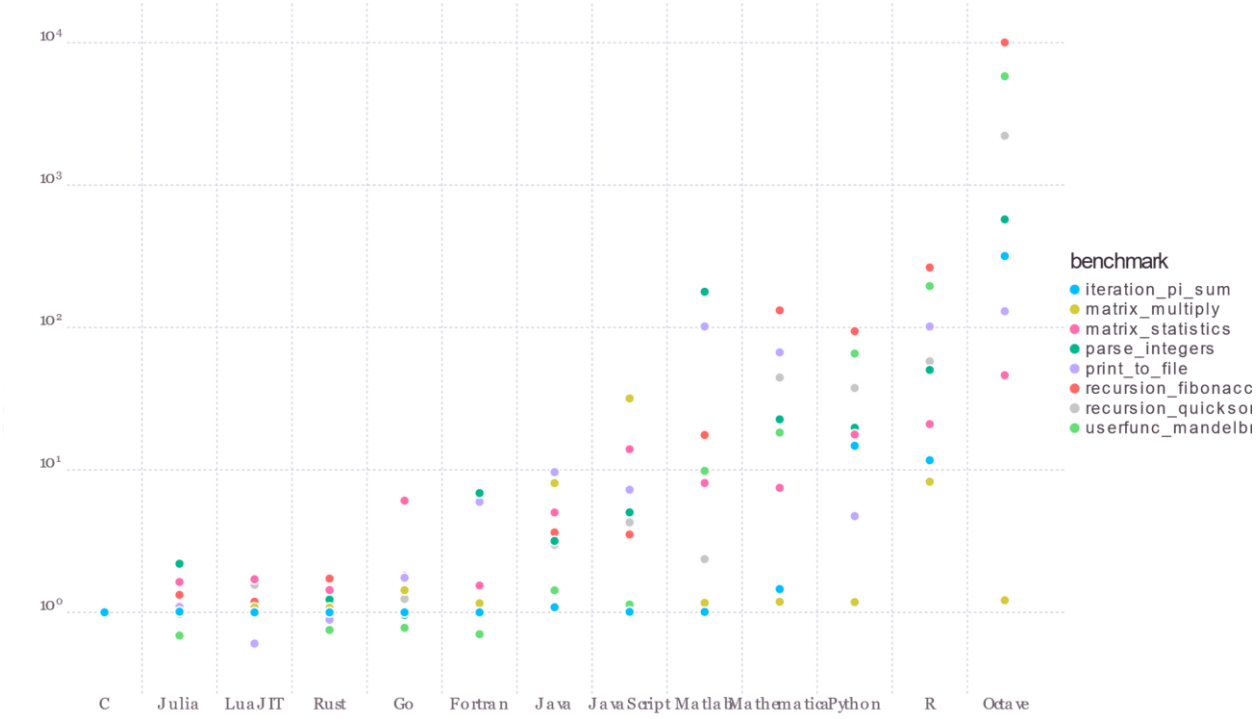

Рисунок 1. *Оценка производительности*

Примечание. Полученные данные, показанные выше, были вычислены с помощью Julia v1.0.0, SciLua v1.0.0-b12, Rust 1.27.0, Go 1.9, Java 1.8.017, Javascript V8 6.2.414.54, MATLAB R2018a, Anaconda Python 3.6.3, R 3.5.0 и Octave 4.2.2. C и Fortran компилируются с помощью gcc 7.3.1, беря лучшее время из всех уровней оптимизации (от-O0 до-O3). C, Fortran, Go, Julia, Lua, Python и Octave используют OpenBLAS v0.2.20 для матричных операций;

Mathematica использует Intel® MKL. Реализации Python matrixstatistics и matrix_multiply используют NumPy v1. 14. 0 и OpenBLAS v0.2.20 функции; остальные являются чистыми реализациями Python (исходный код доступен на официальном сайте). Эти результаты микро-бенчмарка были получены на одном ядре (последовательное выполнение) на процессоре Intel® Core™ i7-3960X 3,30 ГГц с 64 ГБ оперативной памяти DDR3 1600 МГц, работающем под управлением openSUSE LEAP 15.0 Linux.

В данном сравнении скорость языка C представлена единицей. Скорости других языков представлены отношением их скорости к скорости языка C.

1. Python. Является фактически языком Data Science. Имеет огромный объём библиотек для работы и обработки данных. Программы на нём легко писать, но они имеют низкую производительность.
2. R. Часто используется в проектах области статистики, где имеет большой набор вспомогательных пакетов. Также имеет большие возможности в визуализации данных. Это однопотоковый язык, что и объясняет неутешительные результаты.
3. MATLAB. Это коммерческий продукт, разрабатываемый компанией «MathWork», специализированный на матричных операциях, отсюда хорошие результаты матричных испытаний. Тем не менее, в среднем, результат положительный.
4. Octave. Язык программирования, созданный как бесплатная альтернатива MATLAB. Главной целью была совместимость, а не производительность. Отсюда такие плохие результаты.
5. Mathematica. Это коммерческий продукт, разрабатываемый компанией «Wolfram Research». У него нет определённой направленности. В своём устройстве использует Python, поэтому показатели схожи.
6. JavaScript и Go. Оба языка программирования основываются на Google V8. Перед запуском они компилируются в собственный машинный код, что увеличивает производительность. Несмотря на хорошие показатели, оба языка предназначены для веб приложений.
7. Fortran, Go, Rust, LuaJIT. По производительности сопоставимы с Julia. Решение выбора какого-либо из них должно основываться на иных качествах и характеристиках языков.

Типизация

Языки программирования традиционно делятся на два лагеря: со статической и динамической типизацией. Языки со статической типизацией предполагают, что каждой переменной должен быть определён тип еще до запуска программы. В то время, как языки с динамической типизацией способны запускаться с переменными неизвестного типа и в процессе работы определять метод обработки переменной.

В связи с этими различиями, статические языки имеют большую производительность, поскольку работают с заранее определёнными типами, но требуют больших трудозатрат для написания кода. Программирование на динамических языках легче чем на статических, но они уступают в производительности.

Julia имеет древовидную систему типов [4], часть которой представлен на рисунке 2. Ветви этого дерева называются абстрактными типами, а листья конкретными [6]. Значения могут принадлежать лишь конкретным типам [7]. Абстрактные типы лишь объединяют конкретные типы. Это нужно для удобства определения типов переменных. Например, если мы работаем с вещественными числами, то мы должны указать принадлежность переменной к типу Real.

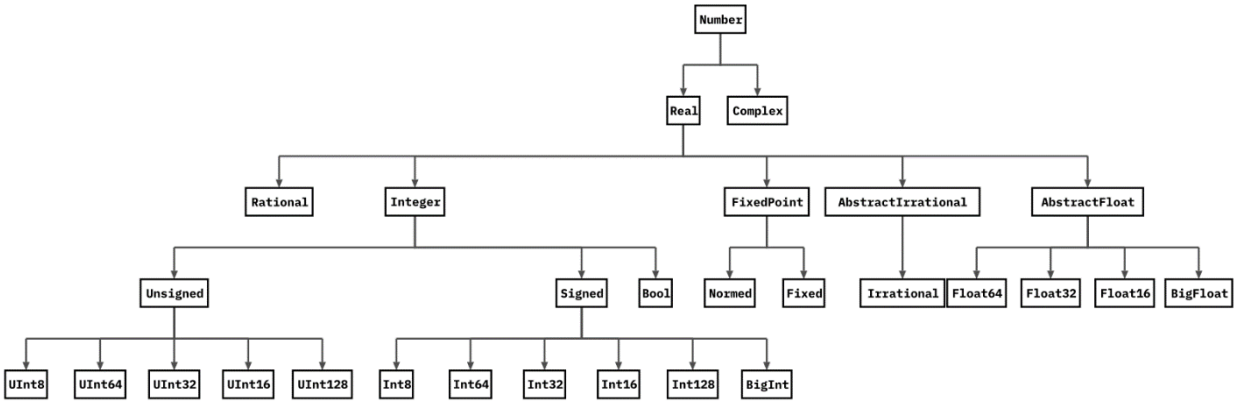

Рисунок 2. *Структура типа данных «Number*

Такая структура системы типов данных позволяет довольно гибко и точно указывать переменным типы данных, что позволяет приблизиться к языкам со статической типизацией, перенимая их производительность [8].

Параллелизм

Очень часто сложные математические задачи решаются на нескольких ядрах одного или нескольких процессоров. Распределённые вычисления сами собой представляют сложную задачу, поэтому выбор подходящего языка программирования очень важен. Язык Julia, как язык для математических вычислений, настроен на параллелизм «из коробки». Пример кода представлен на рисунке 3.

```
julia> using Distributed, SharedArrays

julia> addprocs(2);

julia> a = SharedArray{Float64}(10);

julia> @distributed for i = 1:10
           a[i] = i
       end
Task (runnable) @0x0000000012d7d150

julia> for x in 1:10
         "$(a[x])  " |> print
       end
1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  10.0
```

Рисунок 3. *Пример параллельных вычислений*

В наше время большие объемы вычислений выполняются в облачных средах, где по требованию выделяются нужные ресурсы. Стандарт MPI, который используют для организации работы крупномасштабных параллельных приложений, не отличается особой эластичностью. В ходе вычислений нельзя добавить процессоры, и возможности восстановления после сбоев весьма ограниченны.

В Julia передача сообщений отличается от MPI. Вместо модели «запрос — ответ» используется глобально распределенное адресное пространство [5], что позволяет легко оперировать ссылками на объекты на разных машинах и на ходу добавлять другие ресурсы.

Синтаксис

Создатели постарались сделать синтаксис как можно более сильнее приближенным к MATLAB [1]. Отсюда индексация массивов с единицы и расположение их в памяти по столбцам, а также подавление вывода завершающей точкой с запятой. В то же время Julia не переняла спорные «особенности» MATLAB, такие как символ

комментария, круглые скобки в доступе к элементам массива и так далее [3]. Сравнение синтаксисов приведён ниже (рис. 4).

Julia

```
# Это комментарий в Julia
#=
Многострочный комментарий
=#
if i <= N # Условие
# Действия
else
# Альтернативные действия
end
while i <= N # Условие
# Действия
end
for i = 1:N
# Действия
end
```

MATLAB

```
% Это комментарий в MATLAB
%{
Многострочный комментарий
%}
if i <= N % Условие
% Действия
else
% Альтернативные действия
end
while i <= N # Условие
% Действия
end
for i = 1:N
% Действия
end
```

Рисунок 4. *Сравнение синтаксиса языков Julia и MATLAB*

Сходство синтаксисов языков отнюдь не случайно. Оно свидетельствует о том, что создали Julia намеренно снизили пороговый уровень вхождения в язык именно для пользователей MATLAB [1].

Дополнительные преимущества

1. Удобная документация. Вся документация по языку представлена на официальном сайте docs.julialang.org, содержит множество примеров, также есть возможность искать непосредственно из консоли набора кода [10].
2. Поддержка юникода. Позволяет присваивать переменным названия, содержащие любые символы [6]. Например переменную, содержащую длину волны мы можем назвать напрямую «λ».
3. Макросы и другие возможности метапрограммирования. Это ускоряет оборачивание библиотек на других языках [10].
4. Встроенный пакетный менеджер [6]. Удобное управление пакетами из консоли.
5. Возможность вызова функций языков Python и C [5].
6. Open Source. Julia является open source проектом, это значит что любой человек может получить доступ к исходному коду [11]. Также на язык распространяется бесплатная лицензия MIT.

Выводы

Язык программирования Julia имеет отличные перспективы развития: он имеет высокую производительность, многопоточность представлена «из коробки», простой синтаксис и сложную систему классов. Уже в течении пяти лет он имеет стабильную

тенденцию к развитию: количество библиотек растёт, его сообщество увеличивается. Но из-за его молодости и нераспространённости, он всё еще проигрывает своим конкурентам по количеству библиотек и «коробочных решений».

Сейчас язык Julia подойдёт исследователям лишь как дополнительный инструмент. Но с течением времени Julia будет брать на себя всё большее количество задач, таким образом занимая свою нишу.

Список литературы

1. Антонюк В.А. Язык Julia как инструмент исследователя. Москва: Физический факультет МГУ им. М.В. Ломоносова, 2019.
2. Шиндин А.В. Язык программирования математических вычислений Julia. Базовое руководство. Нижний Новгород: Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2016.
3. Computing in Operations Research using Julia // arxiv.org URL: https://arxiv.org/pdf/1312.1431.pdf (дата обращения: 26.09.2020).
4. Fast Flexible Function Dispatch in Julia // arxiv.org URL: https://arxiv.org/pdf/1808.03370.pdf (дата обращения: 26.09.2020).
5. Ivo Balbaert. Getting Started with Julia. ISBN 9781783284795 изд. Packt Publishing, 2015.
6. Julia Documentation // docs.julialang.org (дата обращения: 26.09.2020).
7. Julia: A Fast Dynamic Language for Technical Computing // arxiv.org URL: https://arxiv.org/pdf/1209.5145.pdf (дата обращения: 26.09.2020).
8. Julia: A Fresh Approach to Numerical Computing // arxiv.org URL: https://arxiv.org/pdf/1411.1607.pdf (дата обращения: 26.09.2020).
9. Julia: dynamism and performance reconciled by design // dl.acm.org URL: https://dl.acm.org/doi/pdf/10.1145/3276490 (дата обращения: 26.09.2020
10. Malcolm Sherrington. Mastering Julia. ISBN 9781783553310 изд. Packt Publishing, 2015.
11. The Julia language // julialang.org (дата обращения: 26.09.2020).

References

1. Antonyuk V. A. the Julia language as a researcher's tool. Moscow: faculty of Physics, Lomonosov Moscow state University, 2019 [in Russian].
2. Shindin A.V. programming language for mathematical calculations Julia. Basic guide. Nizhny Novgorod: Lobachevsky national research Nizhny Novgorod state University, 2016 [in Russian].
3. Computing in Operations Research using Julia // arxiv.org URL: https://arxiv.org/pdf/1312.1431.pdf (date of the application: 26.09.2020).
4. Fast Flexible Function Dispatch in Julia // arxiv.org URL: https://arxiv.org/pdf/1808.03370.pdf (дата обращения: 26.09.2020).
5. Ivo Balbaert. Getting Started with Julia. ISBN 9781783284795. Packt Publishing, 2015.
6. Julia Documentation // docs.julialang.org (date of the application: 26.09.2020).
7. Julia: A Fast Dynamic Language for Technical Computing // arxiv.org URL: https://arxiv.org/pdf/1209.5145.pdf (date of the application: 26.09.2020).
8. Julia: A Fresh Approach to Numerical Computing // arxiv.org URL: https://arxiv.org/pdf/1411.1607.pdf (date of the application: 26.09.2020).
9. Julia: dynamism and performance reconciled by design // dl.acm.org URL: https://dl.acm.org/doi/pdf/10.1145/3276490 (date of the application: 26.09.2020
10. Malcolm Sherrington. Mastering Julia. ISBN 9781783553310. Packt Publishing, 2015.
11. The Julia language // julialang.org (date of the application: 26.09.2020).