phuse

Julia for High-Performance Computing

phuse.global

Contents

Abstract

Julia is a high-level, high-performance, dynamic programming language that was designed from the ground up for numerical and scientific computing. This review paper provides an overview of Julia's features and benefits for high-performance computing, including its fast compilation, dynamic typing, multiple dispatch, and built-in support for parallelism. It also discusses current real-world use cases of Julia and potential case studies for high-performance computing, along with recommended readings and references for further exploration.

1. Background About Julia

Julia is a high-level, high-performance, dynamic programming language that was developed with the goal of combining the ease of use of scripting languages such as Python and MATLAB with the performance of compiled languages such as C and Fortran. It has had many releases since its initial development in 2009.

1.1. Julia Release History

Julia was initially developed in 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah and Alan Edelman, who were motivated by the limitations of existing programming languages in scientific computing and set out to create a free language that was both high-level and fast. The first public release of Julia, version 0.1, was first introduced in 2012 and has since gained popularity in the scientific computing community due to its speed, ease of use and powerful features.

Since then, Julia has undergone several major releases, with new features and improvements added in each release. Version 0.3 was released in August 2014, version 0.6 in June 2017, and both Julia 0.7 and version 1.0 on 8 August 2018. Version 1.0 marked a major milestone for Julia as it was the first stable and production-ready release of the language.

Since the release of version 1.0, Julia has continued to evolve rapidly, with frequent releases of new versions. Versions 1.1, 1.2, 1.3 and 1.4 were released in 2019, while versions 1.5 and 1.6 were released in 2020. Version 1.7 is the latest stable release as of 2022, while versions 1.8 and 1.9 are currently in beta. Julia 2.0 does not yet have a planned release date.

Each release of Julia has introduced new features and improvements to the language, such as improved performance, new syntax, and expanded support for libraries and tools. Julia's development is guided by a strong focus on community involvement, with contributions from a large and active community of users and developers. Julia is open-source software and is licensed under the MIT License, which allows for free and unrestricted use and distribution of the language.

1.2. Julia Key Features

Julia's design is optimised for numerical and scientific computing, which makes it popular in research and academia. Its features can be summarised as follows:

- **Fast:** Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via low level virtual machine (LLVM).
- **Dynamic:** Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.
- **Reproducible:** Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.
- **Composable:** Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns.
- **General:** Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire applications and microservices in Julia.
- **Open source:** Julia is an open-source project, with over 1,000 contributors. It is made available under the MIT License. The source code is available on GitHub.

Julia is compiled to LLVM, a compiler infrastructure that provides high-performance code generation for a wide range of platforms. LLVM was originally developed as part of the LLVM project, which aimed to provide a flexible and reusable compiler framework for programming languages including Julia, Rust and Swift. In Julia's case, LLVM is used to compile Julia programs to efficient native code that can run on a wide range of platforms.

For more information on Julia's features, here are some helpful links:

- The Julia documentation (https://docs.julialang.org/en/v1/) provides a comprehensive guide to Julia's syntax, features and package ecosystem.
- The Julia Language: A Concise Introduction (https://arxiv.org/abs/2008.01451) is a paper that introduces Julia's design and features.
- The Unreasonable Effectiveness of Multiple Dispatch (https://www.youtube.com/watch?v=kc9HwsxE1OY) is a talk by Stefan Karpinski that explains the benefits of Julia's multiple dispatch system.

Julia is a high-performance, dynamic programming language that combines the ease of use of scripting languages with the speed of compiled languages. One of its key features is its speed, which is achieved through just-in-time compilation and support for multi-threading and distributed computing. Another is its dynamic type system, which allows for flexible and expressive programming. Julia also supports multiple dispatch, which makes it easy to express many object-oriented and functional programming patterns. Other features include metaprogramming, asynchronous I/O, reproducible environments, and support for machine learning and data analysis through packages such as Flux.jl and DataFrames.jl. Julia is used by over 10,000 companies and over 1,500 universities and has been applied in fields including machine learning, data science, scientific computing, finance, robotics and materials science.[1,2,3,4,5]

[1] Julia Computing Raises $24M in Series A, Former Snowflake CEO Bob Muglia Joins Board
[2] The Rise of the Julia Programming Language – Is it Worth Learning in 2022? | DataCamp
[3] Why We Use Julia, 10 Years Later
[4] The Julia Programming Language: The History and Uses
[5] What is Julia Used For? 10 Applications of Julia Programming | DataCamp

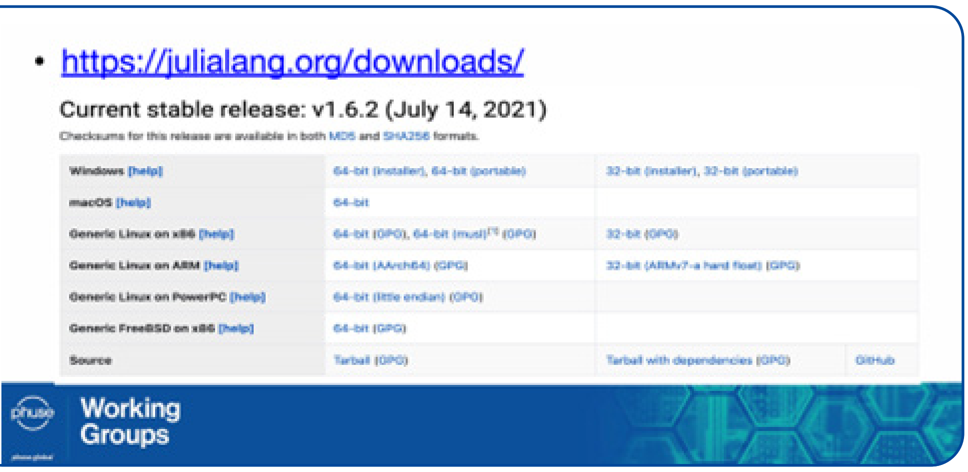
2. Getting Started with Julia

To get started with Julia, download the latest stable release of Julia from the official website (https://julialang.org/downloads/). Julia has a simple and intuitive syntax that is easy to learn for users familiar with programming languages such as Python or MATLAB. Julia's package manager, Pkg, makes it easy to install and manage packages, which are libraries of code that extend Julia's functionality.

2.1. Julia Installation Steps

Here are the main steps for installing Julia and Jupyter:

Step 1: Download and Install Julia



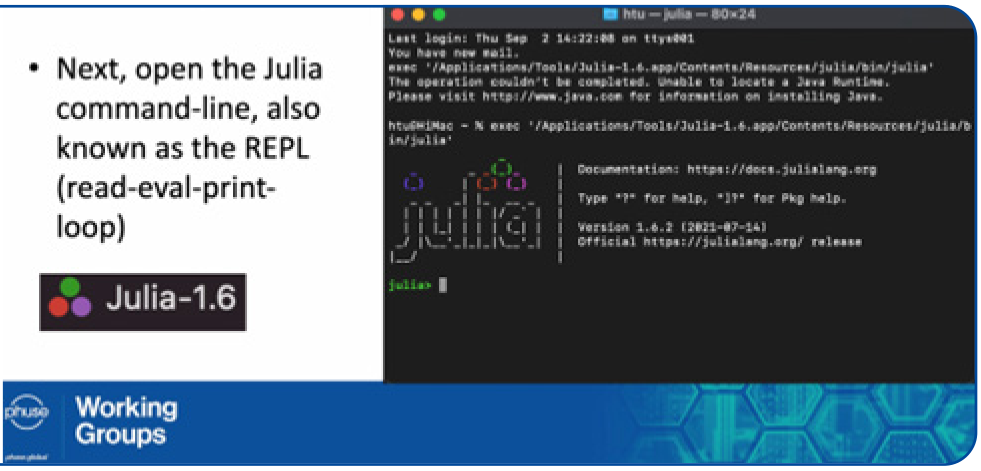
- https://julialang.org/downloads/

Current stable release: v1.6.2 (July 14, 2021)

Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]	64-bit (installer), 64-bit (portable)	32-bit (installer), 32-bit (portable)	
macOS [help]	64-bit		
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl)[1] (GPG)	32-bit (GPG)	
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)	32-bit (ARMv7-a hard float) (GPG)	
Generic Linux on PowerPC [help]	64-bit (little endian) (GPG)		
Generic FreeBSD on x86 [help]	64-bit (GPG)		
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

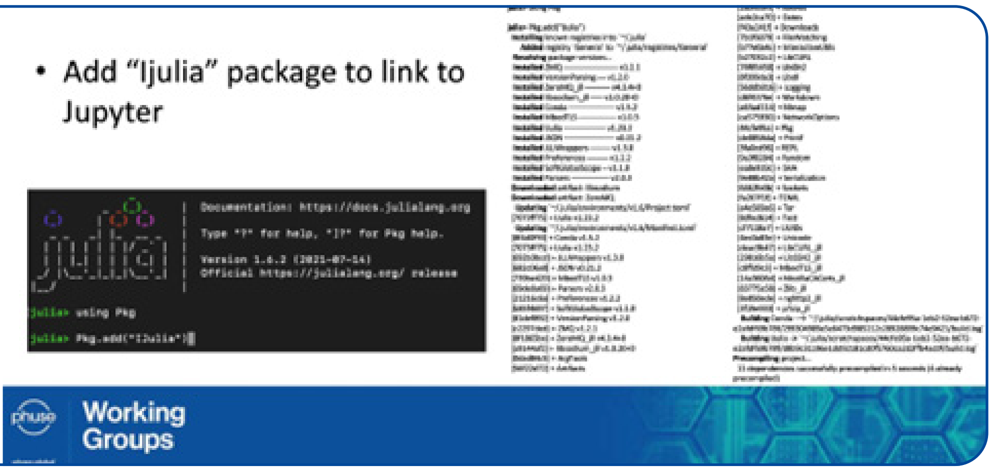
Step 2: Open the Julia Command-Line



- Next, open the Julia command-line, also known as the REPL (read-eval-print-loop)

```
Last login: Thu Sep  2 14:22:08 on ttys001
You have new mail.
exec '/Applications/Tools/Julia-1.6.app/Contents/Resources/julia/bin/julia'
The operation couldn't be completed. Unable to locate a Java Runtime.
Please visit http://www.java.com for information on installing Java.

htu@HiMac ~ % exec '/Applications/Tools/Julia-1.6.app/Contents/Resources/julia/b
in/julia'
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.2 (2021-07-14)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 
```

Step 3: Add Julia to Jupyter Notebook

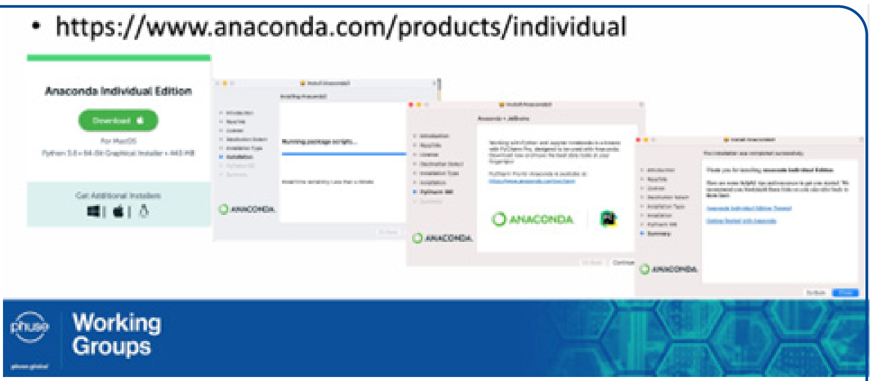


- Add "Ijulia" package to link to Jupyter

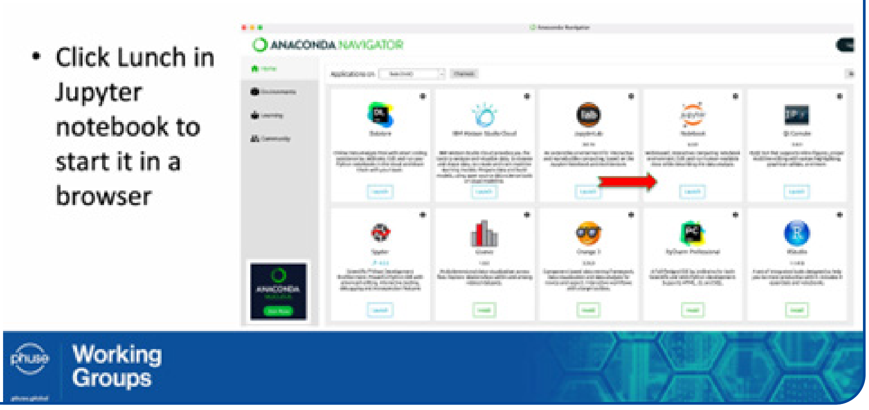
```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.2 (2021-07-14)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> using Pkg

julia> Pkg.add("IJulia")
```

```
julia> using Pkg

julia> Pkg.add("IJulia")
  Installing known registries into `~/.julia`
       Added registry `General` to `~/.julia/registries/General`
   Resolving package versions...
   Installed ZMQ ──────────── v1.2.1
   Installed VersionParsing ── v1.2.0
   Installed ZeroMQ_jll ───── v4.3.4+0
   Installed libsodium_jll ── v1.0.20+0
   Installed Conda ────────── v1.5.2
   Installed MbedTLS ──────── v1.0.3
   Installed IJulia ───────── v1.23.2
   Installed JSON ─────────── v0.21.2
   Installed JLLWrappers ──── v1.3.0
   Installed Preferences ──── v1.2.2
   Installed SoftGlobalScope ─ v1.1.0
   Installed Parsers ──────── v2.0.3
  Downloaded artifact: libsodium
  Downloaded artifact: ZeroMQ
    Updating `~/.julia/environments/v1.6/Project.toml`
  [7073ff75] + IJulia v1.23.2
    Updating `~/.julia/environments/v1.6/Manifest.toml`
  [8f4d0f93] + Conda v1.5.2
  [7073ff75] + IJulia v1.23.2
  [692b3bcd] + JLLWrappers v1.3.0
  [682c06a0] + JSON v0.21.2
  [739be429] + MbedTLS v1.0.3
  [69de0a69] + Parsers v2.0.3
  [21216c6a] + Preferences v1.2.2
  [b85f4697] + SoftGlobalScope v1.1.0
  [81def892] + VersionParsing v1.2.0
  [c2297ded] + ZMQ v1.2.1
  [8f1865be] + ZeroMQ_jll v4.3.4+0
  [a9144af2] + libsodium_jll v1.0.20+0
  [0dad84c5] + ArgTools
  [56f22d72] + Artifacts
  [illegible]
  [ade2ca70] + Dates
  [f43a241f] + Downloads
  [7b1f6079] + FileWatching
  [b77e0a4c] + InteractiveUtils
  [b27032c2] + LibCURL
  [76f85450] + LibGit2
  [8f399da3] + Libdl
  [56ddb016] + Logging
  [d6f4376e] + Markdown
  [a63ad114] + Mmap
  [ca575930] + NetworkOptions
  [44cfe95a] + Pkg
  [de0858da] + Printf
  [3fa0cd96] + REPL
  [9a3f8284] + Random
  [ea8e919c] + SHA
  [9e88b42a] + Serialization
  [6462fe0b] + Sockets
  [fa267f1f] + TOML
  [a4e569a6] + Tar
  [8dfed614] + Test
  [cf7118a7] + UUIDs
  [4ec0a83e] + Unicode
  [deac9b47] + LibCURL_jll
  [29816b5a] + LibSSH2_jll
  [c8ffd9c3] + MbedTLS_jll
  [14a3606d] + MozillaCACerts_jll
  [83775a58] + Zlib_jll
  [8e850ede] + nghttp2_jll
  [3f19e933] + p7zip_jll
    Building Conda ─→ `~/.julia/scratchspaces/[illegible]/build.log`
    Building IJulia → `~/.julia/scratchspaces/[illegible]/build.log`
Precompiling project...
  11 dependencies successfully precompiled in 5 seconds (4 already precompiled)
```

Step 4: Download and Install Anaconda and Products

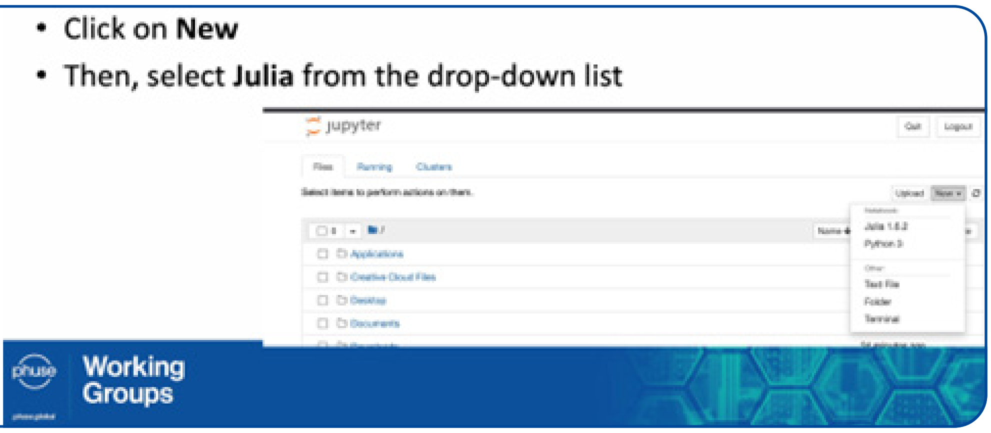


- https://www.anaconda.com/products/individual



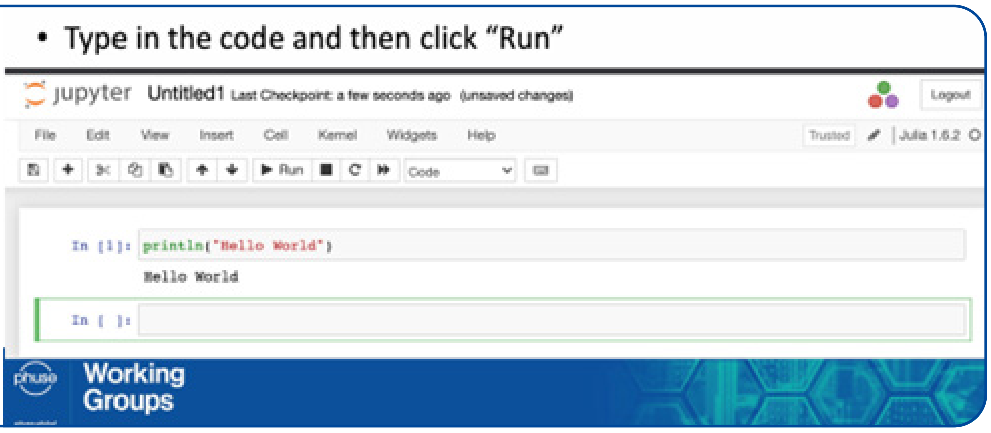
- Click Lunch in Jupyter notebook to start it in a browser

Step 5: Create a New Notebook



- Click on **New**
- Then, select **Julia** from the drop-down list

Step 6: Write Your Code



- Type in the code and then click "Run"

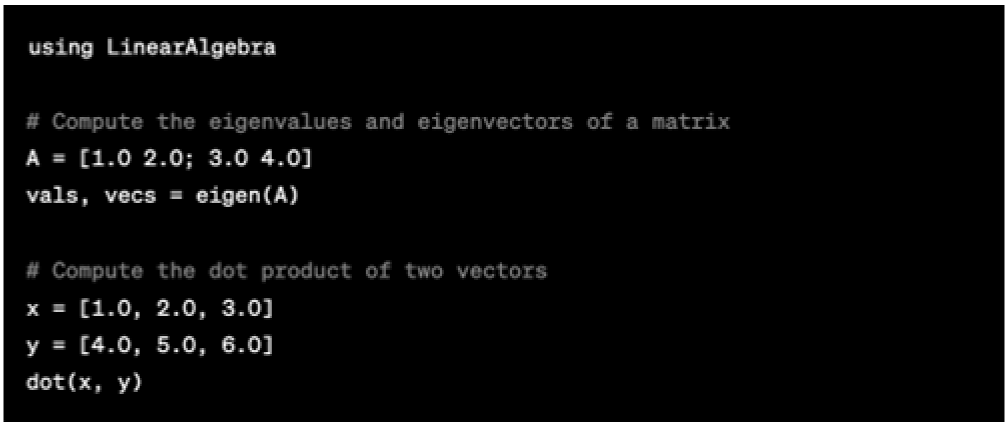
```
In [1]: println("Hello World")
        Hello World

In [ ]:
```

A more recent alternative to Jupyter Notebook is Quarto. See https://quarto.org/. This is especially useful for analysis and reporting and in general for publishing reproducible, production-quality articles, presentations, websites, blog posts and books in HTML, PDF, MS Word, ePub, and more.

2.2. Recommended Packages

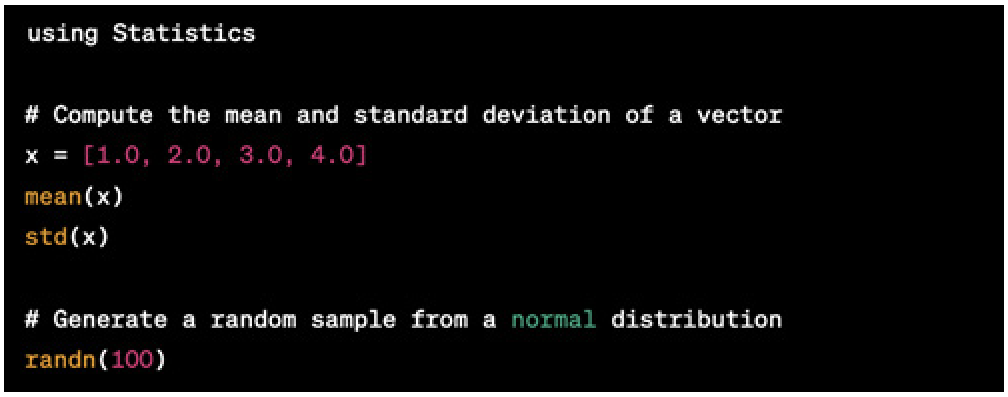
Some recommended packages for high-performance computing include:

• **LinearAlgebra:** provides a wide range of matrix and vector operations, including eigenvalue decomposition and singular value decomposition. Here are some examples



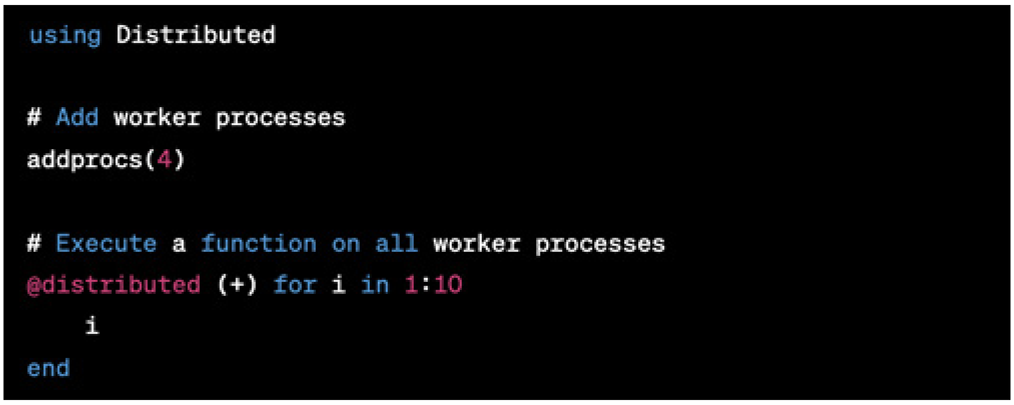
```julia
using LinearAlgebra

# Compute the eigenvalues and eigenvectors of a matrix
A = [1.0 2.0; 3.0 4.0]
vals, vecs = eigen(A)

# Compute the dot product of two vectors
x = [1.0, 2.0, 3.0]
y = [4.0, 5.0, 6.0]
dot(x, y)
```

• **Statistics:** provides statistical functions for data analysis and modelling, including probability distributions, hypothesis testing and regression analysis. This package is built into Julia, so no additional installation is required. Here are some example usages:



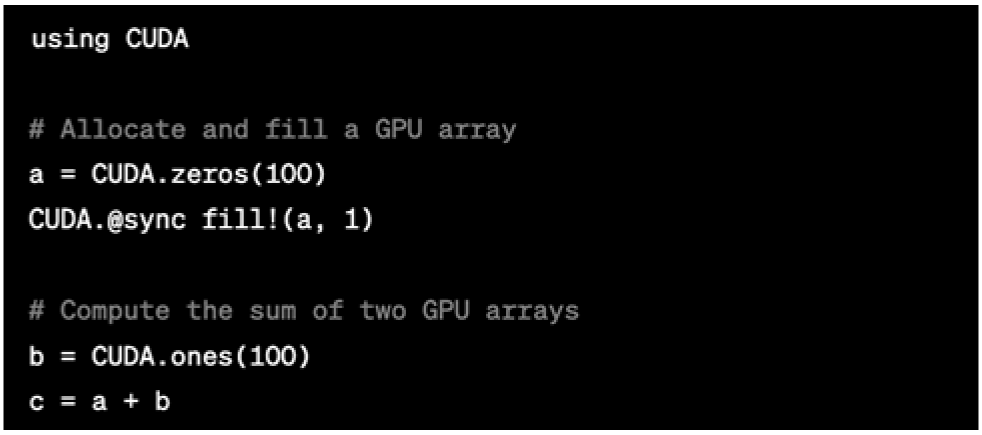
```julia
using Statistics

# Compute the mean and standard deviation of a vector
x = [1.0, 2.0, 3.0, 4.0]
mean(x)
std(x)

# Generate a random sample from a normal distribution
randn(100)
```

• **Distributed:** provides support for distributed computing, allowing Julia code to run on multiple processors or nodes.



```julia
using Distributed

# Add worker processes
addprocs(4)

# Execute a function on all worker processes
@distributed (+) for i in 1:10
    i
end
```

• **CUDA (Compute Unified Device Architecture):** provides support for GPU computing using the CUDA programming model.



```
using CUDA

# Allocate and fill a GPU array
a = CUDA.zeros(100)
CUDA.@sync fill!(a, 1)

# Compute the sum of two GPU arrays
b = CUDA.ones(100)
c = a + b
```

CUDA is a parallel computing platform and application programming interface (API) developed by NVIDIA that allows developers to harness the power of NVIDIA GPUs for high-performance computing tasks. CUDA provides a C/C++-like language for programming NVIDIA GPUs, as well as a set of libraries and tools for optimising and debugging GPU code.

The power of CUDA comes from the massively parallel architecture of NVIDIA GPUs. While CPUs are designed to perform a small number of tasks quickly and efficiently, GPUs are designed to perform a large number of simpler tasks in parallel. This makes GPUs well suited for high-performance computing tasks such as scientific simulations, machine learning and data processing.

By using CUDA, developers can write code that runs on the GPU in parallel, taking advantage of the many cores available on modern GPUs. This can lead to significant speed-ups compared to running the same code on a CPU. For example, deep learning algorithms that might take days or weeks to train on a CPU can be trained in hours, or even minutes, on a GPU using CUDA.

In Julia, the CUDA.jl package provides support for using CUDA in Julia code. With CUDA.jl, developers can write GPU-accelerated code in Julia using familiar syntax and idioms, while still taking advantage of the power of CUDA. This makes it easy for Julia developers to harness the power of NVIDIA GPUs for high-performance computing tasks.

Other useful packages for data analysis include:

- **DataFrames:** This package provides a set of tools for working with tabular data in Julia. Its design and functionality are similar to those of pandas (in Python) and R.
- **Distributions:** This package provides a large collection of probabilistic distributions and related functions.
- **MixedModels:** This package defines linear mixed models (LinearMixedModel) and generalised linear mixed models.
- **GLM:** This package is for linear and generalised linear models.
- **MultivariateStats:** This package is for multivariate statistics and data analysis (e.g. PCA).
- **Gadfly:** This package is a system for plotting and visualisation written in Julia and is based largely on ggplot2 for R and the grammar of graphics.
- **JuliaHealth:** A set of Julia packages for use in medicine, healthcare, public health and biomedical research.
- **MLJ (Machine Learning in Julia):** This is a toolbox written in Julia that provides a common interface and meta-algorithms for selecting, tuning, evaluating, composing and comparing about 200 machine learning models written in Julia and other languages.

There are more packages for very common statistical usages, such as Random and others which we could not list and discuss here.

3. Current Real-World Use Cases

Julia is increasingly being used in real-world applications. It is currently one of the 12 languages to know for data science, along with Python, R, SQL, C/C++, Java, JavaScript, Go, Scala, Swift, MATLAB and SAS. Julia[6][1] is fast and flexible, and this has made it a go-to language for processing big data. Let's look at some general use cases and then some specific use cases within life science.

3.1. Real-World Use Cases in General

Julia is becoming an increasingly popular choice for scientific computing and data analysis due to its speed, ease of use and powerful features. Here are some real-world use cases that highlight the capabilities of Julia in various fields:

- **Machine Learning:** Julia's speed and flexibility make it well suited for developing machine learning models. For example, the Flux.jl package (https://fluxml.ai/) provides a suite of tools for deep learning and other machine learning tasks in Julia. Julia has also been used for developing models for natural language processing, image recognition and other applications. You can see examples of Julia in data science for machine learning, data visualisation and data manipulation (https://juliadatascience.io/).

- **Finance:** Julia's speed and support for distributed computing make it well suited for financial modelling and data analysis. For example, the QuantEcon.jl package (https://quantecon.org/quantecon-jl/) provides a suite of tools for quantitative economics and finance in Julia. Julia has also been used for developing trading algorithms, risk management models, portfolio optimisation and trading strategies (https://juliapackages.com/p/strategems).

- **Robotics:** Julia has been used for developing control systems for robots and other autonomous systems. For example, the RobotOS.jl package (https://github.com/jdlangs/RobotOS.jl) provides a set of tools for robot control and simulation in Julia. Julia has also been used for developing reinforcement learning models for robotic control.

- **Aerospace Engineering:** Julia's speed and support for distributed computing make it well suited for large-scale simulations in aerospace engineering. For example, the ModelingToolkit.jl package (https://mtk.sciml.ai/stable/) provides a set of tools for modelling and simulating aerospace systems in Julia. Julia has also been used for developing control systems for drones and other unmanned aerial vehicles. NASA's Jet Propulsion Laboratory uses Julia to simulate spacecraft trajectories and optimise mission designs (https://www.youtube.com/watch?v=iJr_lU7_7Go).

- **Materials Science:** Julia has been used for modelling and simulating materials properties and behaviour. For example, the MaterialsProject.jl package (https://github.com/JuliaFEM/Materials.jl) provides a suite of tools for materials research and analysis in Julia. Julia has also been used for developing machine learning models for predicting materials properties.

- **Scientific Computing:** Julia is used in scientific computing for numerical simulations, computational physics and computational chemistry (https://docs.juliahub.com/General/JuliaDB/stable/). Julia's support for distributed computing and parallelism make it well suited for high-performance data processing tasks such as real-time data analytics and large-scale data processing. For example, the JuliaDB.jl package (https://github.com/JuliaData/JuliaDB.jl) provides a fast and flexible database system for Julia.

- **Scientific Simulations:** Julia is a popular choice for scientific simulations such as computational fluid dynamics, astrophysics and quantum mechanics. For example, the QuantumOptics.jl package (https://qojulia.org/) provides a toolkit for simulating quantum optics systems using Julia.

These are just a few examples of the many ways that Julia is being used in scientific computing and data analysis. Its speed, flexibility and ease of use make it a powerful tool for tackling a wide range of computational tasks.

3.2. Real-World Use Cases in Life Science

Julia is becoming an increasingly popular choice for scientific computing due to its speed, ease of use and powerful features. Here are some real-world use cases that highlight the capabilities of Julia in life science:

- **Modeling Infectious Diseases:** Julia has been used to model the spread of infectious diseases, such as COVID-19. For example, the COVIDAnalytics.jl package (https://blog.jcharistech.com/2020/04/20/data-analysis-of-covid19-using-julia/) provides a toolkit for analysing COVID-19 data using Julia. Researchers have also used Julia to model the transmission dynamics of other infectious diseases, such as Ebola.

- **Genomics:** Julia is well suited for genomics data analysis due to its speed and support for distributed computing. For example, the BioJulia project (https://biojulia.net/) provides a suite of packages for genomics data analysis in Julia. Julia has also been used for genome assembly and annotation, as well as for analysing large-scale genomic datasets.

- **Drug Discovery:** Julia's speed and flexibility make it well suited for drug discovery tasks, such as virtual screening and molecular docking. For example, the MolSim.jl package (https://juliamolsim.github.io/) provides a suite of tools for molecular simulation and drug discovery in Julia. Julia has also been used to develop machine learning models for predicting drug efficacy and toxicity.

- **Neuroscience:** Julia has been used for modelling neural systems, analysing neuroimaging data and developing machine learning models for brain-computer interfaces. For example, the NeuroJulia project (https://github.com/JuliaNeuroscience) provides a collection of packages for neuroscience research in Julia. Julia has also been used for developing deep learning models for brain-inspired computing.

[6] 12 Top Data Science Programming Languages 2023 | Built In

These are just a few examples of the many ways that Julia is being used in life science research. Its speed, flexibility and ease of use make it a powerful tool for tackling a wide range of scientific computing tasks. There are several companies that have started using Julia to perform various tasks. Here are some examples:

- Pfizer: Pfizer used Julia to accelerate the simulation of treatments for heart failure by 175 times using models running in Julia.[7] [2]
- AstraZeneca: AstraZeneca uses Bayesian neural networks with the Flux.jl and Turing.jl Julia packages to predict drug toxicity. [2]
- Julia Computing: Julia Computing is a company founded by the creators of Julia that offers commercial support and services for Julia. Julia Computing has raised $24 million in Series A funding and is used by over 10,000 companies and over 1,500 universities. [8] [3]

The popularity and adoption of Julia in industry and academia continues to grow, with companies and organisations of all sizes recognising the benefits of using Julia for high-performance computing tasks. [9, 10, 11] [4][5][6]

4. Potential Tasks for High-Performance Computing

Julia's speed, flexibility and ease of use make it well suited for a wide range of high-performance computing tasks in life science. Here are some potential case studies for using Julia in life science:

- **Genome Assembly:** Julia's support for distributed computing and parallel processing makes it well suited for genome assembly tasks, which involve assembling long DNA sequences from shorter reads. With Julia, researchers can efficiently process and analyse large-scale genomics datasets to generate accurate genome assemblies.

- **Protein Folding:** Protein folding is a complex computational problem that involves predicting the three-dimensional structure of a protein based on its amino acid sequence. Julia's speed and support for machine learning make it well suited for developing accurate and efficient models for protein folding, which could have implications for drug discovery and other areas of research.

- **Disease Diagnosis:** Julia's speed and support for machine learning make it well suited for developing models for disease diagnosis based on patient data, such as medical imaging or genomic data. With Julia, researchers can efficiently analyse and process large-scale datasets to develop accurate and reliable models for disease diagnosis.

- **Drug Discovery:** Julia's speed and flexibility make it well suited for virtual screening and molecular docking tasks in drug discovery, which involves screening large databases of compounds to identify potential drug candidates. With Julia, researchers can efficiently screen large-scale databases and develop accurate and efficient models for predicting drug efficacy and toxicity.

- **Neuroscience:** Julia's support for distributed computing and machine learning makes it well suited for developing models of neural systems and analysing large-scale neuroimaging datasets. With Julia, researchers can efficiently process and analyse large-scale neuroimaging datasets to develop accurate models of neural systems, which could have implications for understanding brain function and developing new therapies for neurological disorders.

These are just a few examples of the many potential case studies for using Julia in life science research. Its speed, flexibility and ease of use make it a powerful tool for tackling a wide range of computational tasks in life science.

5. Recommended Julia Readings and References

For further exploration of Julia and high-performance computing, here are some recommended readings and references:

- The Julia Documentation (https://docs.julialang.org/en/v1/) provides a comprehensive guide to Julia's syntax, features and package ecosystem.
- JuliaAcademy (https://juliaacademy.com/) provides online courses and tutorials on Julia for beginners and advanced users.
- The JuliaCon Conference (https://juliacon.org/) is an annual conference that showcases the latest developments and applications of Julia.
- "Julia: A Fresh Approach to Numerical Computing" (https://arxiv.org/abs/1411.1607) is a paper by the creators of Julia that provides an overview of the language's design and features.
- "The Unreasonable Effectiveness of Multiple Dispatch" (https://www.youtube.com/watch?v=kc9HwsxE1OY) is a talk by Stefan Karpinski that explains the benefits of Julia's multiple dispatch system.
- The Julia Programming Language Channel on YouTube: https://www.youtube.com/c/TheJuliaLanguage.

6. Conclusion

In conclusion, Julia is a powerful and versatile programming language that offers many features and benefits for high-performance computing. With its fast compilation, dynamic typing, multiple dispatch and built-in support for parallelism, Julia is becoming an increasingly popular choice for scientific simulations, machine learning and high-performance data processing. This review paper introduces Julia's features and benefits for high-performance computing, along with practical guidance on getting started with Julia, real-world use cases, potential case studies, and recommended readings and references.

[7] Why We Use Julia, 10 Years Later
[8] Julia Computing Raises $24M in Series A, Former Snowflake CEO Bob Muglia Joins Board
[9] Why We Use Julia, 10 Years Later
[10] The Julia Programming Language: The History and Uses
[11] What is Julia Used For? 10 Applications of Julia Programming | DataCamp

7. References

[1] "Julia's power has especially made it a go-to option for big data analysis in the private sector and scientific research. The language can be found in noteworthy projects for climate modelling, weather forecasting and astronomical surveys, among others. Its wide-ranging interoperability – compatible with everything from Python to old-school Fortran – and its continued boost from MIT, also make Julia a likely long-term contender."
12 Top Data Science Programming Languages to Know | Built In https://www.datacamp.com/blog/the-rise-of-julia-is-it-worth-learning-in-2022

[2] "It's not just aerospace engineering – pharmaceutical companies are developing with Julia as well. Pfizer accelerated the simulation of treatments for heart failure 175 times using models running in Julia. AstraZeneca uses Bayesian neural networks with the Flux.jl and Turing.jl Julia packages to predict drug toxicity."
https://juliahub.com/case-studies/astra-zeneca/

[3] "Julia solves the 'two-language problem' by providing high performance and ease of use in a single language. Julia is used by over 10,000 companies and over 1,500 universities. Julia's creators won..."
https://www.prnewswire.com/news-releases/julia-computing-raises-24m-in-series-a-former-snowflake-ceo-bob-muglia-joins-board-884269978.html

[4] "It is taught at hundreds of universities and entire companies are being formed that build their software stacks on Julia. From personalised medicine to climate modelling, novel materials and even space mission planning – everywhere you look, the Julia community is pushing the boundaries of human discovery."
https://julialang.org/blog/2022/02/10years/

[5] "The Julia language was first introduced in 2012 by a team of developers led by Jeff Bezanson, Stefan Karpinski, Viral B. Shah and Alan Edelman. Since its inception, Julia has been used in various fields, including machine learning, data science and scientific computing."
https://leftronic.com/blog/julia-programming-language/

[6] "One of the popular packages is JuliaFin, which specialises in areas such as asset management, risk management, algorithmic trading, backtesting, and other areas of computational finance, including modelling financial contracts." 9. Robotics Roboticists from MIT use Julia to program robots."
https://www.datacamp.com/blog/what-is-julia-used-for

[7] Roesch, E., Greener, J.G., MacLean, A.L., et al. (2023). Julia for biologists. Nat Methods 20, 655–664. https://doi.org/10.1038/s41592-023-01832-z

[8] Gao, K., Mei, G., Piccialli, F., Cuomo, S., Tu, J., & Huo, Z. (2020). Julia language in machine learning: Algorithms, applications, and open issues. Computer Science Review 37, 100254. 10.1016/j.cosrev.2020.100254

8. Acknowledgements

This white paper was developed by the Julia Initiative for High-Performance Computing project in the Data Visualisation & Open Source Technology (DVOST) Working Group. Thanks to the following key members, who participated in the initial discussion and in drafting this white paper:

- Hanming Tu (htu@ashandainc.com)
- Melvin Munsaka (melvin.munsaka@abbvie.com)
- Chris Hurley (churley@mmsholdings.com)
- Patrick Hannon (phannon@mmsholdings.com)
- Jeff Mills (millsjf@ucmail.uc.edu)