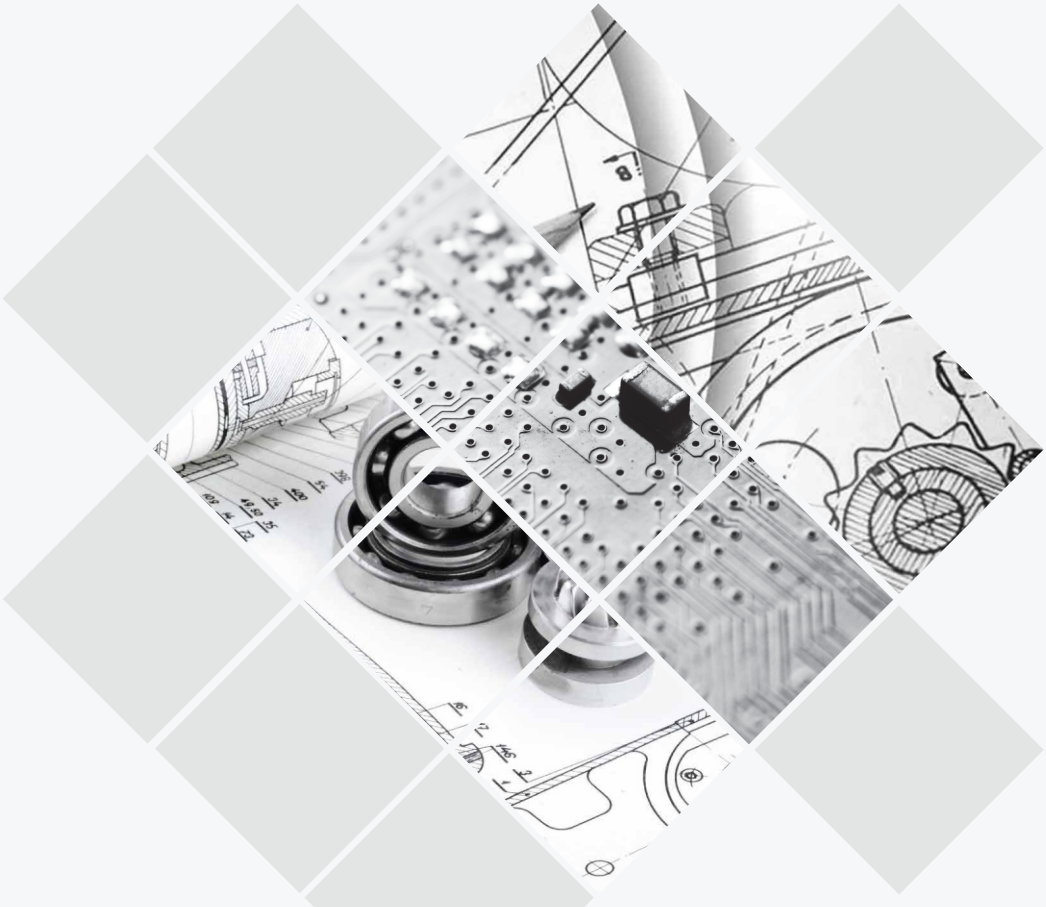
QUEST

BORN TO ENGINEER

Julia Awareness for Scientific Computing

Scientific Computing is an essential part in multiple disciplinary fields of modern engineering; it needs extraordinary computational capabilities to solve complex issues arising in day to day development and application of models and simulation. There are many supporting tools and one such newly evolving tool is Julia.



Velleshala Sudheer

QuEST Global

contents

1. ABSTRACT

Today's scientific world is in need of high technical computing capabilities to solve complex engineering applications. Depending on complexity, each application may involve usage of combination of two or more dynamic platforms like MATLAB, R, C and C++, etc., to achieve the desired functionality and solve the complex problem. So there is always a need to carry out the whole functionality within one platform. Julia, a newly emerging fast technical computing dynamic language, can cater for all the technical computational requirements. Julia can, not only call other C functions but also builds its required libraries and packages in its own language. Moreover Julia is an open-source language.

2. INTRODUCTION

Scientific or technical computing is very much vital in many complex engineering applications, applied math and science. It has a huge scope with numerous computing languages. Among them, FORTRAN, the most conventional Static Programming Language, which was used in numeric and scientific computing language and in converting a high level programming language to a lower level. FORTRAN was a dominant language with remarkable high performance in many engineering areas. C can also be considered as a static language. Both deliver high performance for complex engineering needs. Other computing languages that evolved over the years include MATLAB, R, Mathematica, Octave, Python and SCILAB. All are written in high level language which, at runtime, executes many common programming behaviors. They are known as Dynamic Programming Languages. These languages don't specify the data type like int, float, doubles. etc., which is in contrast with the Static Programming language. The dynamic programming languages provide high convenience and productivity but at the cost of lacking in performance while static languages hold their possession in handling computationally-exhaustive problems for performance but at the cost of lack of productivity. Different applications require different programming choices or a combination of two or more:

- Statistics → R.
- Linear Algebra → MATLAB.
- String Processing → Perl
- General Programming → Python, Java
- Performance, Control → C, C++, Fortran

As numerical computing requires in-built functionalities from library, internal implementation depends on either using the same or different computing language. Retrofitting is not possible. Consequently, there might be some incompatibilities between the programming choices. So in these situations, there is a need of a new dynamic language that caters to the need of both performance and productivity. Julia can be considered as a substitute for existing technical languages. Julia caters to all the requirements and allows to program in a single langauage.

3. PROBLEM STATEMENT

Both the existing Dynamic and Static Computing Languages cater to many challenging areas of numerical technical computing.

♦Unfortunately both have missed out substantially on performance and productivity and the end user is forced to compromise between convenience and performance.

4. PROPOSED SOLUTION

The solution for fast high level system for technical computing is writing the required libraries in one language. For such a requirement, Julia can be considered as a newly emerging fast Dynamic computing language for technical computing. Julia's base library is largely written in Julia itself, also making use of C, C++ and Fortran libraries extensively. These libraries cater to all the needs linear algebra, Signal Processing, String Processing and random number generation.
Many fields, such as mechanical engineering, scientific computing, linear algebra, and computational biology, have massive potential for parallelization and a good fit with Julia's strengths. Julia is very much similar to other technical environments.

Julia provides a
- ► Compiler
- ► Distributed parallel execution
- ► Numerical accuracy
- ► An extensive mathematical function library

4.1. FEATURES OF JULIA

- ► Multiple Dispatch: It's a feature of OOPS, where a function can be dynamically dispatched based on runtime of more than one argument.
- ► Good performance achieved using type inference, approaching that of statically compiled languages like C.
- ► Julia can call C functions directly.
- ► Julia is designed for parallelism and distributed computation.
- ► Julia allows co-routines.
- ► User defined types in Julia are as fast and compact as built-ins.
- ► Built-in package manager is available.
- ► Julia has Powerful shell-like capabilities for managing other processes.
- ► It is free and open source, licensed under MIT (Massachusetts Institute of Technology).
- ► Julia makes use of LLVM (Low-Level virtual Machine) based just-in-time compilation i.e., Dynamic translation.
- ► Provided LISP (LISt Processor)-like macros and other meta programming abilities.
- ► Julia provides a wide range of packages catering to numerous engineering applications.

4.2. MULTIPLE DISPATCH

Generally functions are named based on their purpose. Sometimes these functions may be named similarly, even though the functionality is different. So calling the desired functions becomes puzzled. Multiple dispatch is a process in which a suitable method among several function implementations is dispatched based on run-time type. Multiple dispatch dynamically dispatches or invokes to the implementing method based on the number and datatype of arguments. Basically it is a prime characteristic of Object Oriented Programming.

♦ Dispatch can be dynamic and static. Static dispatch is the implementation of polymorphic operation at compile time. Multiple dispatch are very much useful in object oriented applications like GUI programming.

4.3. TYPE INFERENCE

Automatic deduction of type of a variable or value is known as Type Inference. This ability to automatically infer type from the program makes the programming tasks easier. In dynamically typed languages the type is known at run time. Whereas, in statically typed language, the type of an expression is known only at compile time.

4.4. JUST-IN-TIME COMPILATION

Compilation at the time of execution is just-in time compilation and is also called as dynamic translation. This is in contrast to the conventional procedure of compilation prior to execution. It involves translation to machine code. JIT compilation can yield faster execution than static compilation. JIT compilation is applicable particularly to dynamic capacities. It ensures faster execution of code. Common implementation of JIT compilation includes ahead of time compilation to byte code and then JIT compilation to machine code. JIT compiler compiles the byte code dynamically into machine language. Hence, the execution is faster and is preserved for further use and excludes recompilation. Bytecode is obtained from source code and interpreted prior to JIT compilation.

4.5. LOW LEVEL VIRTUAL MACHINE

Compilation system involves interpretation, compilation and execution. Interpretation and compilation forms the Intermediate levels. LLVM provides intermediate levels for the complete compilation system which will be converted and linked into machine-dependent assembly code. It is compiler infrastructure designed as a set of reusable libraries with well-defined interfaces.

5. APPLICABLE AREAS

Many engineering areas require efficient computational platforms for solving complex engineering problems. In order to cater to the requirement, numerous computing languages have evolved. Both static and dynamic languages have been serving these needs for years. These languages have provided accurate results with high convenience efficiently but unfortunately, each have missed out in terms of performance and productivity. Consequently, there was a need for dependency on two or more languages for accomplishing a single application. Julia is one such language which caters to all the requirements and allows the programmer to work in a single langauage. Julia has a very efficient capability, i.e., it packages. These packages are developed in Julia script itself, which makes the execution faster. Julia covers numerous engineering areas with its functionalities. Some of them are listed below.

► **Signal Processing:**

♦ DSP.jl- Package provides a number of common DSP routines in Julia. Signal processing

functions include Filtering, Spectrograms/Periodograms, Window functions

► **Image Processing:**

♦ Images.jl - An image processing library forJulia.
♦Testimages.jl - Testimages like mandrill.
♦Other useful packages- Color.jl, FixedPointNumbers.jl, ImageView.jl.
♦Functions include imread, show, properties, data, copyproperties, colordim, colorspace, pixelspacing

► **Database:**
♦DataFrames.jl- Read in delimited files into Julia. It also supports reading in gzippped files.
♦ODBC.jl - provides functionality to connect to a database using a Data Source Name (DSN). Functions include readtable, writetable, ODBC.connect, query, advancedconnect.

► **Statistical:**
♦Statsbase.jl - Basic statistics for Julia.
♦Distributions.jl - A Julia package for probability distributions and associated functions.

Functions include:

♦Moments (e.g mean, variance, skewness, and kurtosis), entropy, and other properties.
♦Probability density/mass functions (pdf) and their logarithm (logpdf).
♦Moment generating functions and characteristic functions.
♦Sampling from population or from a distribution.
♦Maximum likelihood estimation.

► **MultivariateStats.jl**

Functions include:

♦Linear Least Square Regression
♦Ridge Regression
♦Data Whitening
♦Principal Components Analysis (PCA)
♦Canonical Correlation Analysis (CCA)
♦Classical Multidimensional Scaling (MDS)
♦Linear Discriminant Analysis (LDA)
♦Multiclass LDA
♦Independent Component Analysis (ICA), Fast ICA.

Julia has a rich package facility as listed above, which can cater to every technical need. Some of the applicable areas are as follows:

♦It is a serious platform for big data applications because of Julia's elegance, power and thriving community [3].
♦Grid Simulation solver provided by IBFS.jl enables grid computing facility. Grid is a distributed system with non-interactive workloads that involve a large number of files.

- ♦Julia can be compiled and built on several ARMv6 or ARMv7 / Cortex A15 Samsung Chromebooks running Ubuntu Linux under Crouton and Raspberry Pi systems. Some are successful and some failed with technical issues.
- ♦Marketing vendors can estimate their business statistics based on the rate at which their online subscribers/customers visit their site and wanting to use their products. This application involves handling of huge data.
- ♦Suitable for applications involving communication between various types of data acquisition and control equipment like DNP3 (Distributed Network Protocol).
- ♦Segregating different type of data based on their characteristics and features. For ex: guessing the age of a person based on the name, as naming style changes between generations.

6. CASE STUDY

Case study 1

Few cases studies have been implemented in Julia and performance analysis is compared with other static and dynamic languages like C++ and MATLAB respectively. The case studies are illustrated below. As an initial example, multiplication code with larger iterations was developed in both MATLAB and Julia. The comparative tabular form showing execution time taken by Julia and MATLAB for different multiplication tables is as shown below:

Multiplication Table	MATLAB	Julia
5x20	0.002873	0.006679681
7x20	0.002969	0.007105129
9x20	0.00312	0.007048684
10x20	0.0040155	0.009385538
100x20	0.004911	0.010239118
1000x20	0.034069	0.00752786
2000x20	0.064765	0.009928864
5000x20	0.153551	0.009693871

Table 1. Mulitplication Table comparing Julia with Matlab.

The comparative graphical analysis for execution time for Julia and MATLAB is shown below. From the graph, it can be clearly inferred that even though the iterations are increasing rapidly, the execution time for Julia is very less when compared to MATLAB.

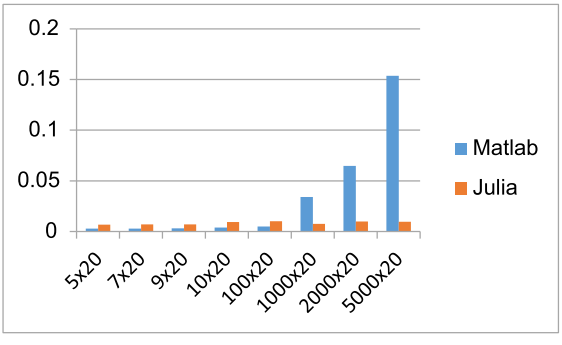


Fig 1. Comparision graph between Julia and Matlab for Multiplication case study.

Case study 2

Refrigeration case was considered as another example, which tries to compare the performance. Julia performance is compared with MATLAB and C++ by evaluating some parameters like Cooling, Power, EER and Pressure of the cylinder. The comparison tabular form is as shown.

sim_end_time	Time_Julia	Time_Cpp	Time_Matlab
5	24.1973418	3.813	484.389
10	50.46536503	7.717	1006.852
15	76.74330158	11.345	1495.6088
20	103.1850586	15.142	1998.7265
30	154.8051001	22.782	2993.6448

Table 1. Refrigeration test case comparing Julia with Matlab and C++.

The comparative graphical analysis for execution time for Julia, C++ and MATLAB is as shown. It can be inferred that Julia is about 20X faster than MATLAB and C++ is 6X faster than Julia.

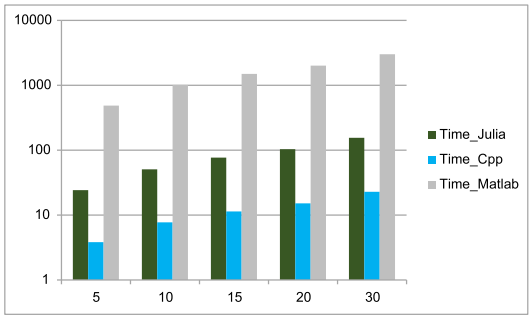
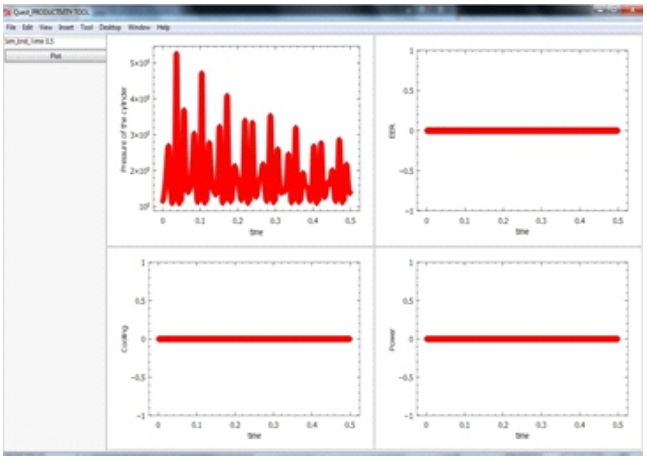


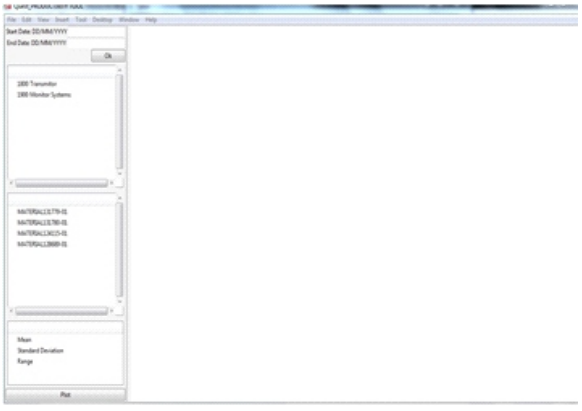
Fig 2. Comparision graph between Julia, C++ and Matlab for refrigeration case study.

The same case study is extended with addition effort and implemented in a Graphical User Interface. The figure below shows user controls like edit box, button and graph, which exemplify the GUI capability of Julia.



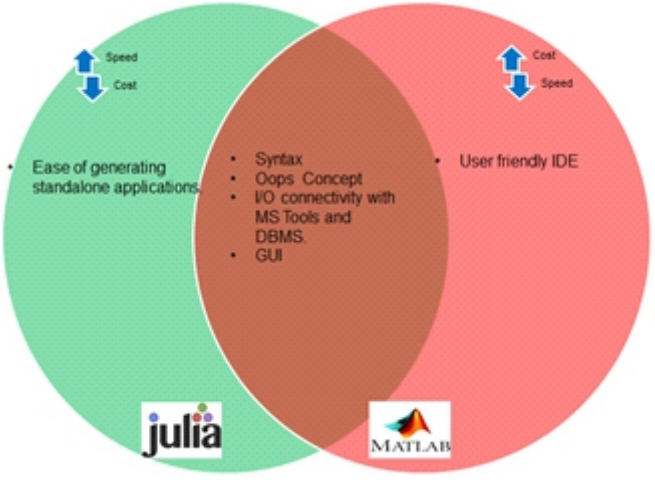
Case Study 3

Another important case study is the Contribution Margin (CM) Tool. The portion of sales revenue that is not spent by variable costs but contributes to the exposure of fixed costs is Contribution Margin. Contribution Margin analysis is a measure of operating leverage. GUI is created for implementing the CM TOOL in Julia. This tool is useful in analysis of revenue depending on incoming and outgoing products in an organization. The figure below exemplifies how analysis is carried out with the help of graphical analysis. The front end view of CM Tool is also shown.



Case study 4

Comparison between Julia and MATLAB is as shown below.



A similar kind of tabular comparison is as shown.

Julia	Matlab
JULIA is high performance dynamic programming language for technical computing.	MATLAB is high level interpreted language optimized for Matrix/array operations and contains useful features.
It is Open source. It is licensed under the MIT License	It is Licensed and not open source.
JULIA has no proper IDE. It does not have all the abilities, which therefore makes the work progress slow.	Simple but powerful MATLAB IDE, it provides excellent user friendly environment for managing programs, editing files, saving workspace, command history, debugging and more.
SIMULINK is not available and GUI programming is possible with appropriate packages.	GUI and SIMULINK come along with toolboxes.
All the functional definitions like plotting etc., are obtained through Packages.	All the advance features like image processing, curve fitting functionalities are obtained through additional toolboxes.
JULIA cares for datatype. Need to declare and initialize variables and classes with size. Or else it throws error.	MATLAB doesn't care for data type, and therefore so free from hassles that arise from data type conversions. No need to declare and initialize variables and classes.
In JULIA, LLVM (Low Level Virtual Machine) provides middle layers of compiler system, i.e., intermediate form, which will be converted and linked into machine-dependent assembly code. LLVM provides compiler infrastructure designed as a set of libraries.	MATLAB follows the hierarchy of compiling, assembling, linking, loading. It's a toolbox infrastructure designed as a set of libraries.
No need of standalone application as JULIA is an open source.	For generating standalone applications, MATLAB requires compiler toolbox. i.e., running an application without MATLAB installed.
Multiple dispatch provide flexibility of function overloading. Designed for parallelism and distributed computation.	Parallel processing available in MATLAB

7. CONCLUSION

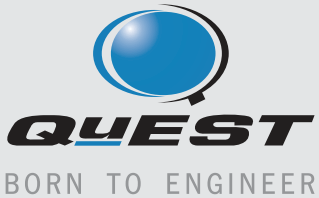
For complex engineering computations, Julia's rich package can cater to various industrial needs. It overcomes the need for relying on other languages for acquiring different functionalities. Various experiments are carried out implementing different application in Julia and other static and dynamic languages like C++ and MATLAB. The inference from the above experiment is, Julia is about 20X faster than MATLAB and C++ is 6X faster than Julia.

8. REFERENCES

1. https://github.com/svaksha/Julia.jl/blob/master/HPC.md
2. https://github.com/JuliaLang/julia/blob/master/README.arm.md
3. http://istc-bigdata.org/index.php/open-big-data-computing-with-julia/

9. AUTHOR PROFILE

Velleshala Sudheer is a Senior Engineer working in the Power Generation DC at QuEST Embedded and Engineering Software. He is an expert in areas like speech and audio processing, satellite tracking and aeronautics. His industrial experience along with his expertise in Julia, MATLAB and DSP environment has made this paper possible.
Sudheer holds a bachelor's degree in Electronics and Communication Engineering, a master's degree with emphasis in Signal Processing and Diploma in Embedded systems.



BORN TO ENGINEER

www.quest-global.com